Plataforma FPGA para Robótica Móvil

por

Juan Carlos Moctezuma Eugenio

Tesis sometida como requisito parcial
para obtener el grado de

MAESTRO EN CIENCIAS EN LA ESPECIALIDAD DE CIENCIAS DE LA COMPUTACIÓN

en el

Instituto Nacional de Astrofísica, Óptica y Electrónica
INAOE
Febrero 2009
Tonantzintla, Puebla

Supervisada por:

Dr. Miguel Octavio Arias Estrada

Resumen

En este trabajo de tesis se presenta el diseño e implementación de una plataforma de control y procesamiento basada en FPGA para aplicaciones de Robótica Móvil. El robot de prueba usado para validación del sistema está basado en un robot móvil programable de la compañía iRobot llamado Create. También se añaden otros elementos como sonares, sensores infrarrojos y brazo manipulador para aumentar la funcionalidad del robot. El diseño de la plataforma FPGA está basado en el concepto de sistemas empotrados, así que ofrece una arquitectura hardware que incluye: procesador, buses, memoria y periféricos (e.g. co-procesadores, sensores, brazo robot, controladores, etc.), todo dentro de un mismo *chip*, por lo que disminuye el costo, tamaño y consumo de potencia del sistema. Así mismo permite la flexibilidad de agregar o modificar nuevos componentes hardware mediante la reconfigurabilidad del FPGA. Además, el trabajo incluye la parte software conformada por un sistema operativo (SO) basado en Linux que junto con un conjunto de bibliotecas que permiten manipular y controlar los componentes hardware de la plataforma FPGA. Todas las funciones están en lenguaje C y se encuentran organizadas en diferentes niveles de abstracción, por ejemplo hay funciones que se encuentran en un lenguaje de fácil uso para los programadores de robots.

La finalidad principal de esta plataforma es mostrar las ventajas que ofrecen los dispositivos FPGA para diseñar plataformas robóticas, ésto se logra mediante una serie de experimentos realizados en un ambiente de prueba. Dentro de estas ventajas, principalmente se tiene el poder de paralelismo y la flexibidad que manejan los FPGAs. Por otro lado, la plataforma propuesta tiene la propiedad de ser escalable para incorporar nuevos elementos hardware, ya sea sensores, actuadores u otro tipo de dispositivo hardware de uso común para aplicaciones en robótica móvil. Muchos de éstos sensores/actuadores necesitan algún tipo de interfaz o de algún preprocesamiento, el FPGA ofrece soluciones flexibles para tratar estas necesidades. Además, la plataforma FPGA permite que la arquitectura software quede abierta para incrementar la funcionalidad de la misma, como por ejemplo la creación de mapas, algoritmos de visión, cinématica de brazos manipuladores, etc.

Abstract

This thesis presents the design and the implementation of a FPGA-based robotic platform for applications on Mobile Robotics. A test robot was used for validation and it is based on a commercial programmable mobile robot called Create from iRobot company. Also test robot includes other components like sonars, infrared sensors and a robotic arm. These additional components are used to increase platform's functionality. In fact, this platform is a complete embedded system. Hardware side includes processor, buses, memory and peripherals (e.g. co-processors, sensors, robotic arm, controllers, etc.), all these components inside of the same chip, therefore the system achieves a reduction in size, power consumption and cost. And allows to add or to modify new hardware components through FPGA reconfigurability. With respect to software side, it includes a Linux-based Operating System (OS) with a set of libraries that perform different funcionalities and manipulate all components on FPGA platform, all these functions are written in C code and they are organized in different levels of abstraction, e.g. there are functions easy-understanding for robotic programmers.

The main purpose of this FPGA-based platform is to show the advantages achieved when FPGAs are used to implement robotic platforms, advantages such as flexibility of implementing different hardware arquitectures or different embedded Linux-based OS, or power of paralelism to accelerate data processing, among others. Another advantage is scalability, this platform allows the addition of different kind of hardware component, protocol bus or custom digital logic. Furthermore software architecture is open to increase its functionality, for example by creating navigation maps, vision algorithms, robotic arm kinematics, etc.

Agradecimientos

Al Consejo Nacional de Ciencia y Tecnología (CONACYT) por el apoyo económico otorgado a través de la beca para estudios de maestría.

Al Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE) y a la Coordinación de Ciencias Computaciones por todas las facilidades y apoyos que me otorgaron durante mis estudios de maestría.

A mi asesor, Dr. Miguel O. Arias Estrada, por su apoyo, consejos y motivación que hicieron posible la realización de esta tesis.

A mis sinodales, Dra. Angélica Muñoz M., Dr. Luis Enrique Sucar S., Dr. Eduardo Morales M., Dr. René Armando Cumplido P., por sus útiles observaciones y comentarios que ayudaron a mejorar esta tesis.

Índice general

Índice de figuras

Lista de Tablas

Capítulo 1

Introducción

1.1. Antecedentes

El principal objetivo de la robótica es la construcción de máquinas capaces de realizar tareas con la flexibilidad, la robustez y la eficiencia que exhiben los seres humanos. Los robots son potencialmente útiles en escenarios peligrosos para el ser humano, tediosos, difíciles o simplemente incómodos. En este sentido, se tienen a los brazos robots que se emplean en las fábricas de automóviles para soldar y pintar, los robots móviles que se envían a Marte o algún lugar en donde el ser humano no puede ingresar, o bien los que se usan para limpiar centrales nucleares, así como también los robots de servicio en los hogares; todos ellos son ejemplos de aplicaciones reales en las cuales se utilizan robots hoy en día.

Dentro del campo de la robótica existe una vertiente llamada robótica móvil, la cual como su nombre sugiere trata con robots móviles, los cuales se pueden definir como una combinación de varios componentes tanto físicos (hardware) como computacionales (software). Un robot móvil puede ser considerado como una colección de subsistemas para locomoción, sensado, cierto nivel de inteligencia y comunicación entre sus componentes (Dudek & Jenkin, 2000).

Existe una gran variedad de robots: con ruedas, con patas, brazos manipuladores, reptantes, humanoides, trepadores, etc. La forma de locomoción es una característica importante en un robot móvil, pero lo que identifica de una forma más fiel a cualquier robot es la forma en como combina a sensores, actuadores y procesadores.

Los sensores miden alguna característica del entorno o propia (e.g. cámaras, sonares, parachoques, infrarrojos, etc.), mientras que los actuadores permiten al robot "hacer algo", es decir, llevar a cabo alguna acción en respuesta a lo sensado, generalmente moverse (e.g. motores). Los procesadores hacen los cómputos necesarios y realizan el enlace lógico entre sensores y actuadores, materializando el comportamiento del robot en el entorno de trabajo (Bräunl, 2006). En este último punto puede haber robots cuyo elemento procesador sea un microprocesador o bien un microcontrolador; ambos teniendo sus ventajas y desventajas dependiendo de la funcionalidad y capacidad de cómputo que se quiera en el robot. En la actualidad ha aparecido una nueva forma para la construcción de plataformas robóticas, y es mediante FPGAs, con estos dispositivos se busca que se puedan usar las ventajas de un microcontrolador y un microprocesador en un mismo dispositivo. También los FPGAs poseen características de paralelismo y reconfiguración, las cuales son buenas opciones para mejorar el poder de cómputo de un robot, por ejemplo, para el procesamiento de video e imágenes en tiempo real. Adicionalmente, el FPGA permite integrar en un solo *chip* procesadores, periféricos, memoria, puertos, etc., reduciendo el espacio, peso, costo y consumo de potencia de la electrónica de control y procesamiento del robot.

Una plataforma robótica es un robot junto con la tarjeta controladora, ésta última funciona como cerebro del sistema. Este tipo de plataformas robóticas se utilizan en la experimentación e investigación en ambientes reales. Las plataformas robóticas se emplean principalmente en la

fase de desarrollo de proyectos de sistemas robotizados, por ejemplo se utilizan para la prueba y validación de arquitecturas de control, o para examinar algoritmos de navegación autónoma o semi-autónoma, todo mediante diferentes tipos de sensores, actuadores y demás dispositivos electrónicos. Así que robots como Pioneer (MobileRobots, Septiembre 2008), Khepera (Kteam, Diciembre 2008), pueden ser consideradas como plataformas robóticas.

Ahora bien, una plataforma de desarrollo o simplemente plataforma, se refiere solamente a la tarjeta controladora, en este sentido, tarjetas como la KoreBot (Kteam, Diciembre 2008), Gumstix (Gumstix, Diciembre 2008), Handy Board (HandyBoard, Diciembre 2008), pueden ser consideradas como plataformas. En este trabajo de tesis se propone una plataforma de desarrollo basada en FPGA, a la que se llama "Plataforma FPGA", con esta plataforma FPGA se busca resaltar las ventajas que se obtienen sobre las plataformas basadas en procesadores o microcontroladores.

Por otro lado, el auge actual de los dispositivos electrónicos se debe al aumento de la capacidad de integración de los fabricantes de circuitos integrados digitales, dispositivos tales como memorias, Micros-Controladores, PLDs (*Programmble Logic Devices*), CPLDs (*Complex Programmable Logic Devices*) y FPGAs (*Field Programmable Gate Arrays*). Pero dicho aumento no hubiera sido posible sin un cambio profundo en los procedimientos de diseño, que tuvieron que pasar de ser manuales a estar basados en herramientas que consisten en complejos programas de computadora, es decir, programas que utilizan a su vez varias herramientas para poder funcionar, coordinar estas herramientas y hacer que todo funcione bajo un solo programa, es una tarea difícil. En este sentido, deja de ser práctico describir sistemas digitales mediante un esquemático o mediante ecuaciones booleanas, y por ello ha sido necesario el desarrollo de lenguajes de descripción de sistemas digitales llamados HDLs (Lenguajes de Descripción Hardware), entre los cuales los más usados son Verilog y VHDL. Aunado a esto se han desarrollado poderosas herramientas que permiten el diseño de sistemas digitales para varias áreas de aplicación, por ejemplo, la empresa Xilinx (Xilinx, Diciembre 2008) cuenta con varias opciones como Xilinx System Generator©que es una herramienta que trabaja bajo el entorno Simulink de MATLAB©y permite realizar sistemas digitales optimizados para procesamiento digital de señales. Otra herramienta de la misma compañía es Plan Ahead©que permite optimizar los diseños de sistemas digitales en cuanto a restricciones de espacio y tiempo, además de permitir realizar reconfiguración parcial dentro del FPGA. Otra de las herramientas importantes y la usada en este trabajo es EDK (*Embedded Design Kit*), la cual permite realizar sistemas embebidos con un alto grado de abstracción en cuanto al diseño de los componentes, de esta forma, el usuario se enfoca a la funcionalidad de su diseño, más que a su construcción (Xilinx-2, Febrero 2008).

Uno de los dispositivos digitales hardware más poderosos que existen en la actualidad son los FPGAs. Los FPGAs son dispositivos programables de propósito general con gran capacidad de integración, son matrices de compuertas programables en el campo, es decir, la funcionalidad del FPGA es definida por un programa de usuario en lugar de estar definida por el fabricante. También los FPGAs poseen características como paralelismo y reconfigurabilidad que los hacen ser dispositivos interesantes para ser usados en la fabricación de plataformas para robots móviles. Los FPGAs ofrecen ventajas sobre los sistemas basados en microprocesadores y microcontroladores como el paralelismo, reconfiguración y realización de sistemas embebidos en un chip, tal y como se presenta a lo largo de este trabajo.

1.2. Objetivos

El objetivo general de esta tesis es: proponer una plataforma FPGA Hardware/Software básica para aplicaciones en robótica móvil. El concepto "básico" para este trabajo se define en la sección 1.3

Para cumplir este objetivo general, se fijaron los siguientes objetivos particulares:

- Desarrollar una arquitectura hardware basada en FPGA para integrar de forma funcional un robot móvil con sus componentes.
- Desarrollar una arquitectura software que permita controlar los elementos hardware e implementar aplicaciones en un lenguaje de alto nivel.
- Desarrollar una plataforma FPGA que de soporte a tres componentes principales: robot móvil, brazo manipulador y sensores.
- Validar la plataforma con aplicaciones y experimentos que reflejen las ventajas de usar el FPGA.

1.3. Alcances y limitaciones

En este trabajo se realiza una plataforma (i.e. tarjeta controladora/procesadora) para robots móviles basada en FPGAs. La plataforma da soporte a un robot móvil comercial de la compañía iRobot llamado Create©, el cual para su locomoción consta con un arreglo de par diferencial con dos ruedas de tracción y una rueda tipo *caster*. Este robot se usa como base para ser controlado mediante la plataforma FPGA, es decir, se aprovecha la carcasa, la parte mecánica y la parte de control de motores del robot. Pero la plataforma FPGA no ésta sujeta sólo a este tipo de robot, es decir, la plataforma FPGA puede ser independiente del tipo de robot móvil utilizado, siempre y cuando el robot móvil tenga una interfaz a bajo nivel y esa interfaz pueda ser imlementada con la lógica del FPGA, entonces el robot móvil puede ser incorporado y controlado mediante la plataforma FPGA. La plataforma FPGA también cuenta con sensores y actuadores, en concreto cuenta con un brazo manipulador, cinco sonares, una brújula y dos sensores infrarrojos. La incorporación de estos sensores muestra la flexibilidad de la plataforma FPGA para poder incorporar una gran variedad de sensores/actuadores, nuevamente el tipo de interfaz de los sensores/actuadores tiene que ser compatible con los recursos del FPGA, la interfaz en un sensor/actuador comercial para robótica en la mayoría de los casos puede ser incorporado usando la lógica del FPGA.

El alcance principal de este trabajo es realizar una plataforma basada en FPGA para robótica móvil con elementos comunes en este tipo de robots, así como el proporcionar una serie de experimentos que hagan notar que una plataforma basada en FPGAs tiene ciertas ventajas con respecto a las plataformas basadas en microprocesadores o microcontroladores tradicionales. Dentro de estos experimentos se tiene: añadir distintos tipos de sensores a la plataforma con el objetivo de demostrar que no importa la forma de comunicación de los sensores, siempre se podrá integrar la mayoría de los sensores utilizados en robótica. Otro experimento es el probar el beneficio del uso de co-procesadores, aumentando así el poder de cómputo en un sistema, en este caso, se realizan pruebas de procesamiento de imágenes a nivel hardware, aprovechando el paralelismo que ofrecen los FPGAs.

La plataforma propuesta en este trabajo es básica, donde "básica" significa que da soporte sólo para algunos sensores/actuadores típicos de un robot móvil (e.g. sonares, brújula y sensores infrarrojos) y da soporte también para el control de un robot móvil comercial. Para este trabajo los elementos que controla la plataforma FPGA son: robot Create, sonares, sensores infrarrojos (de proximidad y seguidor de línea), brazo manipulador, control de motores (i.e. ruedas del robot) y brújula (*compass*). En cuanto a la parte software, se ofrecen bibliotecas de funciones

para controlar los elementos de la plataforma en un lenguaje que sea fácil de usar para los programadores convencionales de robots, éstas funciones se encuentran divididas en capas de abstracción.

En cuanto al ambiente de prueba, se trabaja en un ambiente estructurado específicamente para validar el funcionamiento de la plataforma FPGA. El tipo de ambiente es interior con ciertos objetos extra para el desarrollo de los experimentos como marcas en el suelo, objetos para ser transportados, bases para sostener objetos, entre otros. Las pruebas que se realizan están orientadas principalmente al correcto funcionamiento de la plataforma más que a probar algoritmos "inteligentes" de navegación, o de evasión de obstáculos, o de cinemática del brazo manipulador, etc. Sin embargo, la plataforma debe facilitar la integración de este tipo de algoritmos como trabajos futuros.

La plataforma FPGA está dividida en dos partes: la arquitectura hardware y la arquitectura software, aunque ambas partes funcionan en conjunto. La parte hardware debe tener toda el hardware necesaria para dar soporte tanto al robot móvil como a los sensores. Mientras que a nivel software, se debe ofrecer toda la programación necesaria y un conjunto de funciones para poder manipular el hardware del robot. Ambas arquitecturas están hechas para que sean "abiertas", es decir, que se puedan añadir más componentes y funcionalidades tanto a nivel hardware como software.

La finalidad de incorporar el brazo manipulador es sólo para dar mayor funcionalidad al robot, no se pretenden explorar nuevos técnicas para el manejo de la cinemática del brazo. El brazo solo es usado para transportar objetos de un lugar a otro en el ambiente de prueba. Las funciones de alto nivel que se ofrecen para el brazo permiten mover cada uno de los motores así como manipular sus parámetros (e.g. torque, ángulo de giro, velocidad de giro, etc.). Sin embargo, la plataforma software esta diseñada para poder programar la cinemática del brazo como un trabajo futuro.

Finalmente con este trabajo se propone una plataforma para robots móviles en donde se aprovechen las ventajas que ofrecen los dispositivos FPGA (e.g. paralelismo, reconfiguración) para incrementar el desempeño del robot. La plataforma FPGA propuesta en este trabajo no ofrece ventajas significativas aparentes, más bien los experimentos realizados y los resutados obtenidos tienen el objetivo de dar un primer paso y dejar indicios claros de las posibilidades que pueden alcanzarse para realizar una plataforma más robusta (i.e. soportar algoritmos de visión e implementar tareas en hardware que ayuden al procesador) y completa (i.e. que sirva para dar soporte a múltiples robots, sensores y actuadores) que muestren el verdadero potencial de usar dispositivos FPGAs.

1.4. Organización de la tesis

En esta tesis se propone una plataforma FPGA, cuya arquitecturas hardware y software están orientadas para aplicaciones en robótica móvil. La tesis se encuentra organizada en seis capítulos.

En el capitulo 1 se presenta una breve introducción al trabajo además de los objetivos, los alcances y las limitaciones. En el capítulo 2 se muestran los fundamentos teóricos de las plataformas existentes en el mercado para robótica móvil, sus características, algunas especificaciones de cómo están construidas y cuales son los principales sensores y actuadores con los que cuentan. Por otro lado en el capítulo 3 se expone brevemente lo que son los sistemas embebidos y los FPGAs y de como estos dos se pueden fusionar para formar sistemas completos en un solo dispositivo. En el capítulo 4 se presenta el diseño de la plataforma FPGA, la plataforma se divide en la parte hardware y la parte software, se ilustran las funcionalidades de cada componente de la plataforma así como las ventajas que ofrece la implementación en FPGA. En el capítulo 5 se muestra la implementación de la plataforma FPGA así como los experimentos y resultados obtenidos, también se definen las características del robot móvil de prueba que se utiliza. En cuanto a los experimentos de validación se exponen algunos experimentos individuales para

verificar que la plataforma funciona correctamente y otros que muestran las ventajas de usar FPGAs. Finalmente en el capítulo 6 se dan a conocer las conclusiones y los trabajos a futuro de la tesis.

Capítulo 2

Plataformas para robótica móvil

2.1. Definiciones

Antes de comenzar a analizar algunas de las plataformas para robótica móvil que existen, es necesario realizar algunas definiciones que se seguirán en este trabajo de tesis y así evitar ambiguedades. Las siguientes definiciones están relacionadas con el término "plataforma", algunos fabricantes e instituciones siguen estas definiciones y por lo que respecta a este trabajo se tomarán de la siguiente manera:

Plataforma/Plataforma de desarrollo. El término plataforma (o plataforma de desarrollo) para este trabajo se refiere a una tarjeta controladora para robots móviles, es decir, una tarjeta que ofrece hardware (e.g. memorias, puertos de expansión, conectores, procesadores, microcontroladores, FPGAs, etc.) y software (e.g. herramientas de programación, ambientes de simulación, bibliotecas de funciones, etc.) que sirve para controlar robots móviles. De esta manera tarjetas como EyeBot, Gumstix, KoreBot, pueden ser consideradas como plataformas. Así que cuando se vea el término plataforma (o plataforma de desarrollo) se debe pensar en una tarjeta controladora, existen plataformas basadas en microprocesadores o en microcontroladores.

Plataforma FPGA. Es una tarjeta controladora/procesadora que funciona como "cerebro" para los robots móviles y que esta basada en un FPGA. Dicha tarjeta tiene al FPGA como elemento principal junto con otros elementos adicionales (e.g. memorias ROM, FLASH, RAM, conectores, botones, leds, pantallas LCD, etc.). Además dentro del FPGA se encuentra implementado un sistema embebido que sirve para realizar tareas de control y procesamiento de datos.

Plataforma robótica. Se considera una plataforma robótica al robot móvil junto con su tarjeta controladora, de esta manera los robots que se venden comercialmente (e.g. Pioneer, PowerBot, Kephera, Surveyor, etc.) pueden ser considerados plataformas robóticas.

Las plataformas robóticas se emplean principalmente en proyectos de investigación, en donde, por ejemplo, se utilizan para la prueba y validación de arquitecturas de control, o para examinar algoritmos de navegación autónoma o semi-autónoma y cuentan con diferentes tipos de sensores, actuadores y demás dispositivos electrónicos.

Una de las razones principales por la que grupos de investigación en robótica optan por desarrollar sus propios prototipos de plataformas robóticas es el poder explorar y proponer nuevos algoritmos y nuevas alternativas arquitecturales a nivel hardware que puedan ser más interesantes (i.e. flexibles, con mayor poder de cómputo, etc.) comparadas con las ya existentes. Es por ésto que varias instituciones universitarias y centros de investigación en todo el mundo han desarrollado diversos robots experimentales.

Ahora bien, existen diferentes niveles de prototipos, un grupo de investigación puede realizar toda la plataforma robótica desde cero, es decir, conseguir los motores, ruedas, carcasa, sensores (incluso construirlos desde la electrónica), etc., e ir construyendo el robot dependiendo de las necesidades del proyecto. Esta forma puede ofrecer un robot totalmente "hecho a la medida" pero difícil y tedioso de realizar. Una forma más directa de construir una plataforma robótica es conseguir partes ya elaboradas (i.e. un robot comercial con las necesidades de locomoción suficientes, sensores especiales para robots y otros aditamentos como brazos manipuladores, cámaras, etc.), e ir construyendo la plataforma uniendo todas estas partes. Esto tiene la ventaja de que se puede construir un robot móvil tan complejo como se requiera, con las partes que se necesiten, que sea altamente modular, expansible y flexible.

Con lo anterior, la ventaja de construir una plataforma robótica propia es principalmente en la flexibilidad y escalabilidad que se obtienen. La flexibilidad se logra cuando se tiene un dispositivo en el cual se pueda implementar una gran variedad de módulos electrónicos, que puedan ser probados una y otra vez en el mismo dispositivo, es decir, dispositivos que permitan implementar una gran variedad de arquitecturas hardware que se adecuen a las necesidades del proyecto. En la parte software también tiene que haber flexibilidad, es decir, poder implementar varios tipos de sistemas operativos de acuerdo a las características que se estén buscando. Por lo que respecta a la escalabilidad, una plataforma hardware debe tener los recursos suficientes para poderle añadir más dispositivos, protocolos de comunicación, sensores, procesadores, etc., sin tener que modificar todo o gran parte del sistema. A nivel software, se espera que la plataforma sea escalable, es decir, tener toda la infraestructura a bajo nivel para poder implementar una arquitectura software más elaborada y que ésta se pueda ampliar para realizar funciones cada vez más complejas, nuevamente sin tener que rediseñar toda la infraestructura software que ya se tiene.

Las plataformas permiten al usuario programar de una manera más fácil sus algoritmos. También una plataforma puede ofrecer distintas opciones a nivel hardware que permitan al usuario enriquecer los recursos con los que cuenta y así poder realizar aplicaciones cada vez más complejas (Muñoz et al., 2006). En la siguiente sección se habla en detalle de las plataformas de desarrollo.

2.2. Plataformas de desarrollo

Las plataformas de desarrollo son las tarjetas controladoras que se pueden adquirir comercialmente o bien realizarlas de forma personalizada, y sirven para controlar el robot móvil.

Las aplicaciones con robots móviles presentan cada vez mayor complejidad y ofrecen mayor funcionalidad. De ahí la importancia de tener toda una infraestructura necesaria para poder solventar toda esa complejidad y funcionalidad que puede tener un robot en específico. En esta parte es donde se da el origen y la diferencia entre la parte hardware y la parte software de una plataforma.

La parte hardware es toda la estructura física de la tarjeta electrónica que controla al robot. Esta electrónica generalmente consta de una tarjeta que contiene dispositivos electrónicos para controlar todos los elementos (i.e. sensores, actuadores, motores, cámaras, etc.) como un conjunto. Dentro de esta tarjeta, la cual podría considerarse como el cerebro del robot, se tienen memorias ROM, RAM, FLASH, convertidores A/D, convertidores de niveles de voltaje, procesadores, microcontroladores, FPGAs, DSPs, etc. Para poder controlar y coordinar todos estos dispositivos se requiere de software.

Por lo que respecta a la parte software de la plataforma, se tiene que el modo en como se programan los robots ha ido evolucionando. Históricamente los robots eran desarrollos únicos, no se producían en serie, y los programas de control se construían empleando directamente los controladores (*drivers*) para acceder a los dispositivos sensoriales y de actuación. El sistema operativo del robot era mínimo, básicamente una colección de controladores con rutinas para leer

datos de los sensores y enviar consignas a los actuadores, invocando directamente las funciones de la biblioteca que ofrecía el fabricante en sus controladores.

Con el asentamiento de los fabricantes, el trabajo de muchos grupos de investigación y el crecimiento en las aplicaciones robóticas en cuanto a complejidad han ido apareciendo plataformas de desarrollo que ofrecen capas de abstracción a nivel software, las cuales simplifican el proceso de programación de aplicaciones robóticas. Las capas de abstracción en una plataforma de desarrollo ofrece acceso más sencillo a sensores y actuadores, suelen incluir un modelo de programación que establece una determinada organización del software y permite manejar la creciente complejidad del código cuando se incrementa la funcionalidad del robot (Cañas et al., 2004; Cañas & Matellán, 2002). El diseñador programa sus aplicaciones robóticas finales sobre estas capas de abstracción, tal y como lo muestra la figura 2.1.

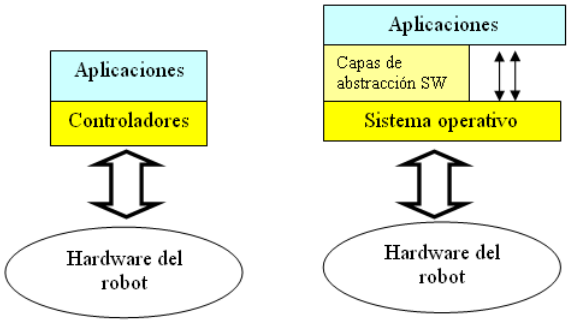


Figura 2.1: Programación de robots sobre controladores específicos de sensores y actuadores (izquierda), sobre capas de abstracción que hacen el manejo del hardware más sencillo (derecha), ya que las capas de abstracción ofrecen bibliotecas de funciones que están en un lenguaje familiar para un programador convencional de robots.

Normalmente el software de las plataformas ofrecen un acceso abstracto y simple a sensores y actuadores. Por ejemplo, si se dispone de un robot Pioneer equipado con un sensor láser STICK, la aplicación puede acceder a sus mediciones a través de las funciones de la plataforma ARIA (MobileRobots, Septiembre 2008) o pedirlas y recogerlas directamente a través del puerto serie.

El acceso abstracto también se ofrece para los actuadores. Por ejemplo, en vez de ofrecer comandos de velocidad para cada una de las dos ruedas en un robot con arreglo par diferencial, se puede ofrecer una sencilla interfaz de V-W (velocidad de tracción y de giro) para la actuación motriz, la cual se encarga de hacer las transformaciones oportunas, de enviar a cada rueda las consignas necesarias para que el robot consiga esas velocidades comandadas de tracción y giro. La uniformidad en las funciones software para el acceso al hardware es el primer paso para favorecer la reutilización de software dentro de la robótica.

En las siguientes dos secciones se habla acerca de las características que debe tener el software para un robot móvil, además se analizan un poco las diferentes opciones que se tienen en cuanto a los sistemas operativos que existen para robots móviles.

2.2.1. Software de robots

La creación de aplicaciones para robots no difiere en lo general de la creación de aplicaciones en otros ámbitos del software. El programador tiene que escribir la aplicación en cierto lenguaje, compilar y enlazar su código con las bibliotecas de la plataforma y/o del sistema operativo, y finalmente ejecutarla en los procesadores a bordo del robot. Aunque la dinámica sea similar, las aplicaciones de robots móviles sí presentan requisitos específicos que condicionan las características de los programas.

Escribir programas para robots móviles es una tarea complicada, ya que los robots son sistemas complejos, requieren de estar sensando continuamente y al mismo tiempo tomar acciones. A continuación se presentan algunas condiciones propias de este tipo de aplicaciones.

- Los programas de robots móviles están directamente conectados a ambientes de prueba, a través de sensores y actuadores. Esto implica que el software debe ser ágil, tomar decisiones con rapidez para controlar los actuadores. Por esta razón, se requiere de actuación en tiempo real[1], si no estricto, al menos blando[2].

- Una aplicación de robots móviles típicamente debe estar pendiente de varias fuentes de actividad y objetivos a la vez. Por ello estas aplicaciones suelen ser concurrentes, en este sentido, los sistemas operativos para robots deberían incorporar mecanismos de multitarea y comunicación de interprocesos.

- Otra cuestión relevante que deben contemplar las aplicaciones es su interfaz gráfica. Aunque la interfaz gráfica no es indispensable para generar y materializar el comportamiento autónomo en el robot, normalmente resulta útil como herramienta de interacción con el usuario, ya sea para visualizar resultados, depurar, seleccionar opciones, etc.

- Los programadores de robots se enfrentan a una creciente heterogeneidad, en distintos sentidos, que dificulta su tarea. En cuanto al hardware, existe una gran diversidad de dispositivos de percepción y de actuación, así como de interfaces con las cuales un programador debe estar familiarizado si quiere escribir programas funcionales para robots. En cuanto al software, las aplicaciones de robots no cuentan con un marco estable, no hay estándares abiertos que propicien la colaboración, la reutilización y la integración de código. Esta falta se debe en parte a la heterogeneidad implícita en robótica, tanto en hardware como en software y en parte a la inmadurez del mercado de robots programables. Aunque hoy en día existen interesantes alternativas e importantes avances como por ejemplo el software *Player/Stage* (PlayerStage, Junio 2008) o el *Robotic Studio de Microsoft* (RoboticStudio, Junio 2008).

2.2.2. Sistemas operativos

La misión principal del sistema operativo (SO) es ofrecer a los programas un acceso básico al hardware del robot, permitir la manipulación y uso de este hardware de manera transparente desde estos programas. Fundamentalmente el SO debe incorporar controladores que den soporte software de bajo nivel a los dispositivos físicos. El sistema operativo suele incluir también soporte para el hardware de comunicaciones (e.g. tarjetas de red inalámbricas, protocolos serial, USB, etc.) y para los elementos de interacción del robot (e.g. botones físicos, interruptores, pantallas, etc).

Gran parte de los robots actuales incluyen sistemas operativos de propósito específico. Sin embargo, desde hace algún tiempo se ha extendido el uso de sistemas operativos de propósito general, empleando en el robot bien computadoras portátiles o computadoras de escritorio empotradas. El motivo principal de la amplia aceptación de los SO de propósito general es sin duda la ventajosa relación prestaciones-precio. Existen por otra parte algunas desventajas al usar este tipo de computadoras, como el tamaño, potencia computacional limitada para ciertas aplicaciones, consumo de potencia, no pueden optimizarse con facilidad con respecto al hardware que usa por ser de uso genérico, entre otras.

[1] Un sistema en tiempo real significa que las tareas que realiza deben producirse dentro de unos intervalos de tiempo determinados por la dinámica del sistema físico que supervisan o controlan.

[2] En un sistema de tiempo real blando se intentan cumplir los plazos de ejecución, pero en caso que alguna tarea finalice un poco tarde no sucede ningún desastre. Puede soportar que algunos tiempos no sean alcanzados.

Los sistemas operativos de propósito general como Linux o Windows se han adentrado en el mundo de los robots. Estos sistemas, además de los controladores hardware, incluyen abstracciones y un conjunto extenso de bibliotecas genéricas, útiles para la programación de robots, tales como las bibliotecas e interfaces para multitarea, comunicaciones e interfaces gráficas.

Hablando en un sentido general, GNU/Linux, es un sistema operativo que ha ganado gran aceptación en la comunidad robótica, por el potente soporte de GNU/Linux para la multitarea, las comunicaciones remotas y las bibliotecas gráficas existentes en ese entorno lo hacen un sistema operativo relevante para las aplicaciones de robots móviles. En particular GNU/Linux proporciona herramientas eficientes y flexibles que permiten abordar con éxito la naturaleza concurrente, distribuida y la necesidad de visualización típicas de los programas para robots (Cañas et al., 2004).

Recientemente se han hecho esfuerzos para desarrollar sistemas operativos enfocados únicamente a robots, un ejemplo de ello es el Robotic Studio de Microsoft (RoboticStudio, Junio 2008), el cual se puede definir como una interfaz visual basada en Windows para el control y simulación de robots móviles, figura 2.2.

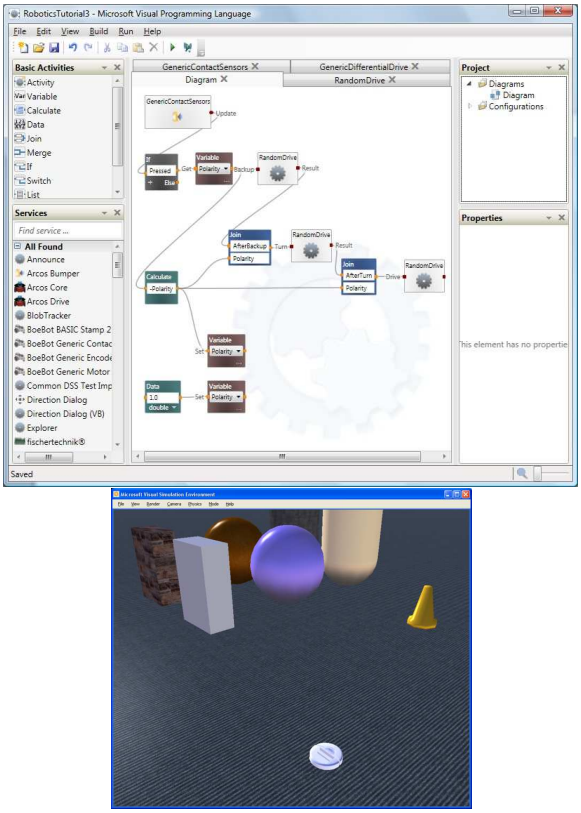


Figura 2.2: *Microsoft Robotic Studio:* ambientes de programación y simulación. Figura tomada de RoboticStudio (Junio 2008)

Robotics Studio está especialmente diseñado para programadores académicos, de entretenimiento o comerciales y soporta una gran variedad de hardware para robots. Dentro de las características principales de esta herramienta están:

- Plataforma de desarrollo para aplicaciones en robótica. Herramienta de programación visual para crear y depurar aplicaciones robóticas. El desarrollador puede interactuar con los robots mediante interfaces basadas en web o en Windows.

- Simulación 3-D. La simulación realista está provista por el motor PhysX de AGEIA. Se posibilita la simulación por software.
- Entorno de ejecución orientado a servicios. El desarrollador puede acceder fácilmente a los sensores y actuadores de los robots, por medio de una librería de implementación de concurrencia basada en .NET. La comunicación estás basada en mensajes, permitiendo la comunicación entre módulos.

Existen otros ejemplos interesante de sistemas operativos para robots son el proyecto *Player/Stage* (PlayerStage, Junio 2008) y *Evolution Robotics* (Evolution, Agosto 2008).

2.3. Plataformas comerciales

En esta sección se habla y se analizan las características principales de algunas de las plataformas comerciales existentes en el mercado. Estas plataformas son algunas de las más utilizadas por académicos, estudiantes y centros de investigación en general ya que ofrecen una buena variedad de prestaciones a nivel hardware y herramientas software para poder programarlas. Además estas tarjetas son una buena alternativa para probar desde algoritmos simples de control hasta algoritmos más complejos como de navegación, seguimiento, visión, entre otros.

2.3.1. Handy Board

Handy Board (HandyBoard, Diciembre 2008) es una tarjeta controladora para robots ampliamente utilizada en el mercado. Esta tarjeta fue desarrollada en el Instituto Tecnológico de Massachusetts (MIT) por Fred G. Martin. La Handy Board es usada principalmente en las universidades y por usuarios para proyectos académicos de mediana complejidad (i.e. control de motores, control de menos de tres sonares, robots pequeños de dimensiones aproximadas de 40×40 cm y soporte para unos cuantos sensores más). En la figura 2.3 se muestra la tarjeta Handy Board y la versión simplificada Handy Cricket de la cual se habla más adelante.

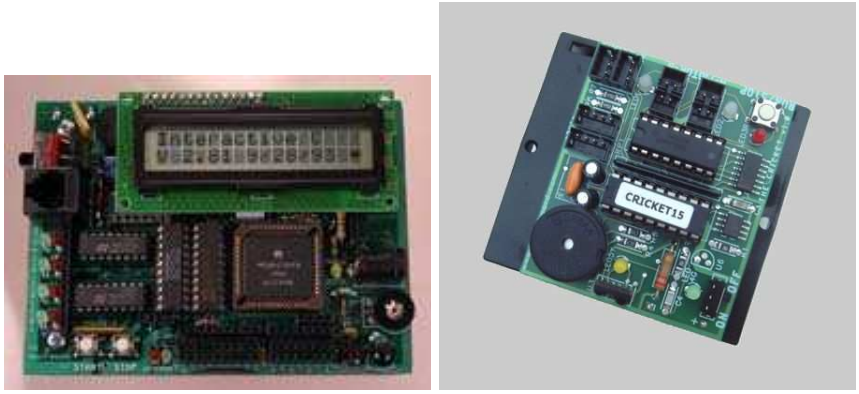


Figura 2.3: Tarjeta Handy Board (izquierda) y una versión simplificada llamada Handy Cricket (derecha). Ambas tarjetas son usadas como controladores de robots, sirven para aplicaciones de mediano nivel en donde no se requiere procesar grandes cantidades de datos, como por ejemplo en algunos algoritmos de visión por computadora como la recuperación en 3D, extracción de movimiento y seguimiento multiobjetivo, entre otros. Figura tomada de HandyBoard (Diciembre 2008)

Dentro de las características importantes de la Handy Board se tiene:

- Microcontrolador Motorola 68HC11 de 8bits y 52 pines.

- Reloj de sistema a 2 MHz
- 32 KB de memoria RAM
- Soporte para motores DC, sensores IR, bus SPI
- Bus de expansión para conectar otras tarjetas

Existen una tarjeta de expansión para la Handy Board, la cual añade otras características, básicamente da mayor soporte para controlar más dispositivos. Dentro de las características principales de esta tarjeta de expansión están: 10 entradas analógicas adicionales, 4 entradas para sensores LEGO, 9 salidas digitales, 6 señales para el control de servos, conector para el sonar Polaroid 6500 y un área de prototipado de propósito general, ésta última es una especie de *protoboard*. La figura 2.4 muestra la tarjeta de expansión para la Handy Board.

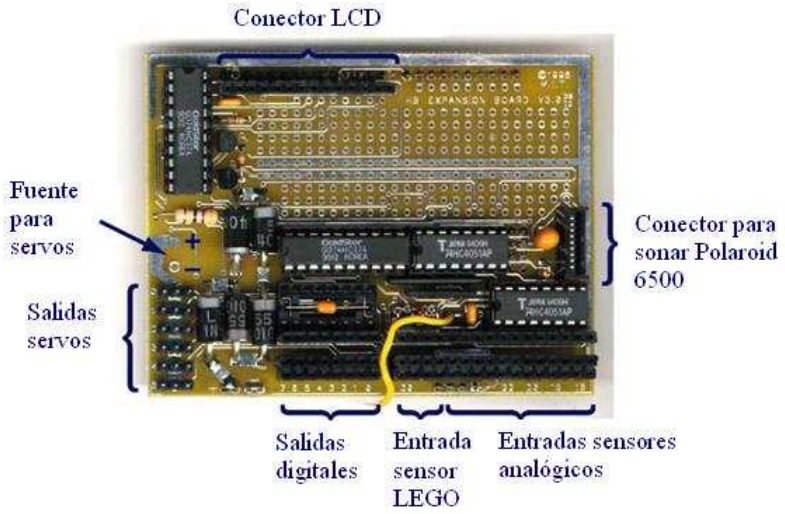


Figura 2.4: Tarjeta de expansión para la Handy Board. La tarjeta de expansión le da mayor funcionalidad a la Handy Board y da soporte a más sensors y actuadores. Figura tomada de HandyBoard (Diciembre 2008)

En cuanto al software para programar la Handy Board se tienen 2 opciones: programar directamente el microcontrolador con las herramientas de Motorola, o bien se puede usar la herramienta Interactive C, ésta última es la más común. Interactive C (IC) es multi-tarea y consiste en un compilador y un módulo de lenguaje máquina que funciona en tiempo de ejecución (*run-time machine language*). IC es un subconjunto del lenguaje C, incluyendo algunas estructuras de control, declaración de variables globales, locales, arreglos, apuntadores, entre otras características.

Existe una versión modificada de la Handy Board llamada Handy Cricket (HandyBoard, Diciembre 2008). Handy cricket es menos potente que la Handy Board en cuanto a desempeño computacional, pero la Handy Cricket tiene las ventajas de ser más pequeña, ligera y barata que la Handy Board. En cuanto al software que usa la Handy Cricket, utiliza el lenguaje "Cricket Logo", el cual es una versión simplificada del lenguaje Logo. A continuación se enlistan algunas de las características más importantes de la tarjeta Handy Cricket:

- Control para motores y luces.
- Comunicación entre tarjetas Crickets mediante mensajes de luz IR

- Aplicaciones de este tipo de tarjeta: control de robots pequeños, juguetes automatizados, lectura de temperatura, etc.
- Bus de expansión por medio del cual se pueden conectar diversas tarjetas para controlar más motores, servos, desplegadores, reelevadores.

2.3.2. Bot Board

La Bot Board II, desarrollada por Lynxmotion (Lynxmotion, Diciembre 2008), es una tarjeta que da soporte a varios microcontroladores *Basic Atom* de la misma compañía. La tarjeta integra varios periféricos para controlar motores y algunos sensores pero que su característica principal es que puede ser controlada hasta por 6 diferentes microcontroladores de 24 ó 28 pines. La figura 2.5 muestra la Bot Board junto con los microcontroladores que soporta. Dentro de las características principales de esta tarjeta están:

- Da soporte para servos y motores DC
- Entradas analógicas y digitales
- Puerto PS2 para conectar un controlador *PlayStation*
- Interfaz de usuario por medio de leds y botones, en donde el usuario se da cuenta del estado de la tarjeta
- Bocina para generar sonidos mediante comandos especiales

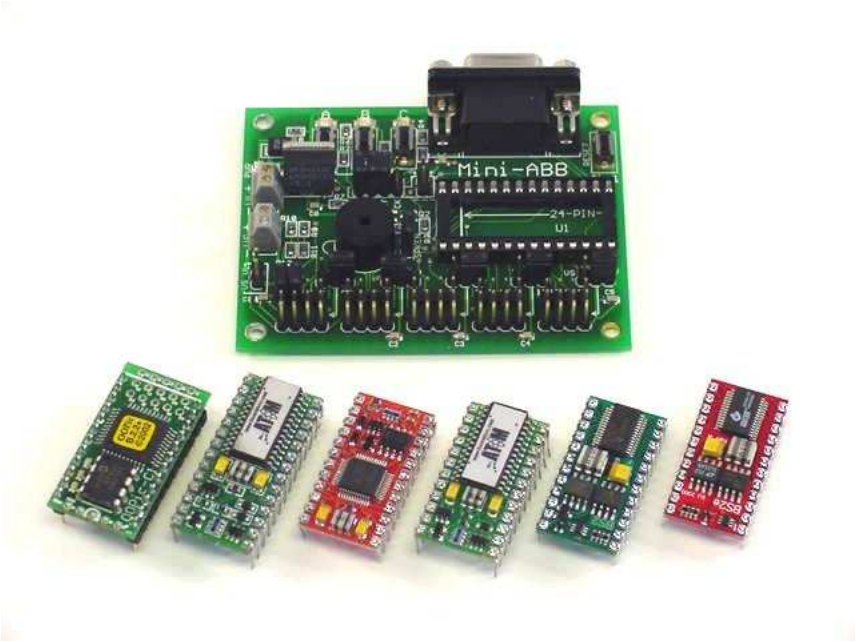


Figura 2.5: Tarjeta Bot Board II y los microcontroladores que pueden ser utilizados en ella. Figura tomada de Lynxmotion (Diciembre 2008)

En cuanto al software que se utiliza para programar la Bot Board, se usa el *AtomPro Language*, el cual es una versión limitada del lenguaje *Basic* y tiene un compilador el cual incluye funciones específicas para controlar los periféricos de la tarjeta.

En cuanto a las tarjetas de expansión, se tienen a la tarjeta SSC-32 que sirve para controlar hasta 32 servos y la tarjeta SaberTooth que sirve para controlar motores DC de hasta 3 A.

2.3.3. KoreBot

KoreBot es una tarjeta de la empresa K-team (Kteam, Diciembre 2008) que fue diseñada para incrementar el desempeño del robot móvil Koala. Esta tarjeta posee un sistema operativo embebido basado en Linux para el fácil desarrollo de aplicaciones, está basada en un microprocesador XSCALE PXA-255 a 400 MHz y tiene las siguientes características:

- Memoria RAM de 64MB y memoria flash de 32 MB.
- Bus I2C, SPI/SSP
- Interfaz UART Bluetooth
- Tarjeta de sonido AC97
- 50 pines E/S de propósito general
- Controlador LCD
- Pines para señales PWM

En la figura 2.6 se muestra la tarjeta KoreBot la cual tiene el tamaño aproximado de una tarjeta de crédito. En cuanto a las tarjetas de expansión, KoreBot tiene la capacidad de poder ser conectada con otros módulos de la misma compañía, como el KoreMotor, KoreSound, KoreIO, los cuales sirven para conectar motores DC, conectar señales de audio y puertos E/S tanto analógicos como digitales respectivamente.

El software que se utiliza es una distribución de Linux 2.4.19 y un programador familiarizado con Linux puede desarrollar aplicaciones para la KoreBot (Kteam, Diciembre 2008), sólo es necesario tener las herramientas de compilación-cruzada (*cross-compilation*) para poder ejecutar el programa en la plataforma KoreBot.

KoreBot es utilizada como "cerebro" para los robots Kephera III de la misma compañía. También se usa en el "Spiderbot" el cual es un robot con patas y algunos otros proyectos en centros de investigación.

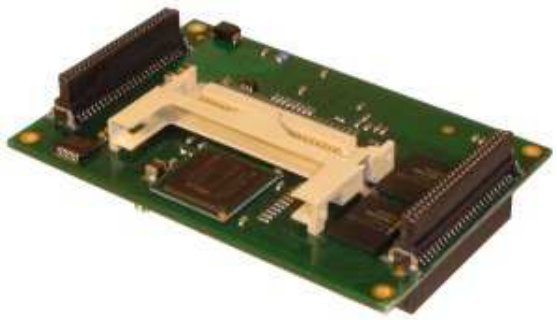


Figura 2.6: Tarjeta KoreBot usada para aplicaciones embebidas de robótica, aunque también se usa para aplicaciones de propósito general. Figura tomada de Kteam (Diciembre 2008)

2.3.4. Gumstix

Gumstix (Gumstix, Diciembre 2008) es la marca registrada de una computadora en una tarjeta basada en el procesador Intel XScale (400 MHz - 600MHz). A la tarjeta se puede acceder por medio del puerto serie, USB, Ethernet o interfaces inalámbricas. La tarjeta corre Linux

empotrado y es utilizada para aplicaciones de propósito general, es simplemente una computadora pequeña que ofrece ciertas prestaciones en cuanto a desempeño y periféricos. La figura 2.7 muestra la tarjeta Gumstix Verdex Pro, la cual es una de las tarjetas más populares dentro de la gama Gumstix. Las características principales de esta tarjeta son:

- 64 MB de RAM y 16 MB de memoria Flash
- adaptador micro-SD
- 3 conectores de 60,80 y 24 pines para conectar tarjetas de expansión
- Interfaz USB
- 3 puertos UART
- hasta 90 pines E/S
- Bus I2C y *Bluetooth*

Figura 2.7: Tarjeta Gumstix utilizada como computadora pequeña de propósito general, es pequeña y liviana. Gumstix es parecida a la KoreBot, sólo que esta última está orientada mayormente para aplicaciones robóticas. Figura tomada de Gumstix (Diciembre 2008)

2.3.5. Plataformas para Visión

Existen algunas plataformas dedicadas a la parte de visión en los robots. La visión es quizá uno de los sensores más importantes en un robot. La visión computacional es una de las áreas de investigación con mucho interés, ya que el proveer del sentido de la vista a un robot móvil puede darle ventajas similares de navegación, localización y percepción que a los animales o personas. Desafortunadamente los algoritmos de visión computacional son muy demandantes computacionalmente (del orden de varios GOPS) para tareas de complejidad mediana y alta, esto ha llevado a los algoritmos a ser simplificados para poder usarlos en robótica móvil. Una alternativa es integrar el procesamiento a la cámara (Rodriguez & Arias, 2002; Arias & Torres, 2001), y así descargar al procesador de tareas de bajo nivel de la visión. Siguiendo esta filosofía se han diseñado algunas plataformas para visión robótica. A continuación se analizan dos plataformas comúnmente usadas para este tipo de aplicaciones.

EyeBot

EyeBot (Eyebot, Diciembre 2008) es una tarjeta controladora para robots, la cual puede ser usada en robots con ruedas, robots caminantes, etc. Se basa en un microcontrolador de 32 bits con un desplegador gráfico y una cámara digital en escala de grises o a color. La cámara es una tarjeta aparte de la Eyebot y se conecta directamente al EyeBot. Esto permite programar potentes aplicaciones sin la necesidad de una computadora grande y pesada que tenga que cargar el robot y además sin tener que sacrificar la propiedad de visión del robot. La figura 2.8 muestra la tarjeta EyeBot y la cámara compatible con este tipo de tarjeta.

Figura 2.8: Tarjeta Eyebot (izquierda) y cámara EyeCam compatible con EyeBot (derecha). La tarjeta EyeBot se usa para aplicaciones sencillas de visión y maneja algunos motores. Figura tomada de Eyebot (Diciembre 2008)

Dentro de las características principales de la tarjeta EyeBot están:

- Programación de aplicaciones que contengan procesamiento de imágenes
- Extensión para controlar mecanismos y sensores propios
- LCD grande (64x128 pixels)
- Programación mediante lenguaje C o lenguaje ensamblador
- 2 puertos seriales
- puertos digitales de E/S
- puertos analógicos
- micrófono y bocina

En cuanto a la cámara EyeCam se tiene que es una cámara a color de 24 bits de resolución por pixel, tiene una baja resolución (80×60) y una mediana resolución (480×640), utiliza un sensor con tecnología CMOS, el cual tiene un mejor rango de brillo que las cámaras CCD, otra característica de la EyeCam es su tamaño ($3 \times 3{,}4$ cm).

El software que se necesita para programar la EyeBot y la EyeCam se llama "RoBIOS", que consiste en un conjunto de bibliotecas que ofrecen diferentes funciones para el procesamiento de imágenes, salida al LCD, configuración de la cámara, semáforos, manejo temporizadores, control de puertos seriales, etc. Dentro de las funciones de procesamiento de imágenes se tiene: el operador laplaciano, operador de sobel, operador de *dithering*, diferencia de imágenes y conversión de imagen RGB a escala de grises. EyeCam es una tarjeta que tiene pocas operaciones de procesamiento de imágenes pero que puede servir para algunas aplicaciones de mediano nivel.

CMUcam

CMUcam (CMUcam, Diciembre 2008) es un dispositivo de bajo costo usado para aplicaciones de visión por computadora. Básicamente la gama de cámaras CMUcams (CMUcam1, CMUcam2 y CMUcam3) se basan en una cámara de video pequeña y un microcontrolador con interfaz serial. La CMUcam fue pensada para ser liviana, pequeña y de bajo costo, además de poder ser usada por otros microcontroladores como tarjeta que se "adhiere" a una plataforma más grande. La CMUcam puede realizar tareas de procesamiento de imágenes y de seguimiento de pequeños objetos dentro de la imagen, la plataforma CMUcam es muy utilizada en los robots móviles para agregarle la capacidad de visión a los robots. La cámara originalmente fue hecha por la universidad de Carnegie-Mellon y después su licencia fue dada para varios fabricantes. La figura 2.9 muestra la tarjeta CMUcam3, la cual tiene un tamaño pequeño (5.5 x 5.7 cm).

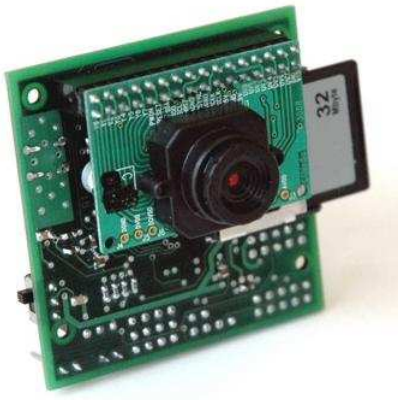


Figura 2.9: CMUcam3, tarjeta basada en un microcontrolador y una cámara para tareas de visión por computadora. La CMUcam3 ofrece mayores ventajas sobre la tarjeta EyeBot en cuanto a las operaciones que puede realizar sobre las imágenes capturadas. Figura tomada de CMUcam (Diciembre 2008)

Dentro de las principales características de esta plataforma se tienen:

- Basada en un sensor CMOS Omnivision
- Resolución de 352x288 RGB color
- Carga imágenes en memoria a una velocidad de 26 FPS
- Entorno de desarrollo de código abierto para Windows y Linux
- Controla hasta 4 servos
- Emulación de la tarjeta predecesora CMUcam2
- Sirve para aplicaciones en Robótica, vigilancia, proyectos académicos, juguetes interactivos, reconocimiento de objetos, etc.

La CMUcam3 puede programarse con varios imágenes *firmware*[3]. En donde dichas imágenes pueden desempeñar diferentes y específicas tareas, por ejemplo se pueden programar aplicaciones de seguimiento, reconocimiento de rostros, vigilancia, etc. Si el usuario desarrolla una

[3] *Firmware* es un bloque de instrucciones de programa para propósitos específicos, y se encuentra grabado en una memoria de tipo no volátil (e.g. ROM), el *firmware* establece la lógica de más bajo nivel para controlar los circuitos electrónicos de un dispositivo.

aplicación propia necesita cargar en memoria Flash dicho programa, esto puede lograrse por medio del puerto serial y usando una utilidad de programación. Dentro de las operaciones que puede realizar la CMUcam3 están: recortar una sección de la imagen, submuestreo de imágenes, funciones de convolución y umbral, entre otras. En general la CMUcam3 ofrece mejor desempeño en cuanto a las operaciones que puede realizar que la EyeCam.

2.4. Resumen de plataformas

Para finalizar en la tabla 2.1 se resumen las características más importantes de las plataformas comerciales. Donde se pueden comparar las diversas opciones que se tienen para trabajar o adquirir alguna de estas tarjetas. Al final del capítulo 5 se vuelve a comparar estas mismas plataformas comerciales con la plataforma FPGA propuesta en este trabajo. Adicionalmente se hace una estimación de la potencia computacional de cada una de las tarjetas para tener un punto adecuado de comparación, y donde esta potencia computacional generalmente no es mencionado abiertamente por el fabricante. La potencia computacional es una estimación basada en el tipo de CPU usado y la frecuencia de operación del reloj. En este sentido la tarjeta Gumstix es la más potente computacionalmente. Las principales limitantes que tienen este tipo de tarjetas comerciales es que algunas fueron hechas solamente para realizar tareas de control y por ello usan microcontroladores, algunas otras como la Gumstix y KoreBot tienen mayor potencia computacional por el uso de microprocesdores, otras como la EyeBot y CMUcam3 hacen procesamiento para tareas de visión pero por sí solas soportan una cantidad limitada de sensores/actuadores, más bien se usan para incorporarse como un sensor "inteligente" a otra plataforma. Esto lleva a tener diversas plataformas comerciales para usos y objetivos diferentes, teniendo que adquirir varias de ellas para diferentes proyectos de robótica. Una opción interesante podría ser tener una plataforma que pueda servir para diferentes usos y objetivos, aparte que ofrezca recursos suficientes para tener una potencia computacional aceptable (como la KoreBot o Gumstix). En el siguiente capítulo se presentan los dispositivos FPGA y el tema de sistemas empotrados en FPGAs, en donde se podrá visualizar que este tipo de dispositivos (FPGA) pueden contrarestar las limitantes de las plataformas comerciales.

Característica	**HandyBoard** uC	Handy **Cricket** uC	**Bot Board II** + **procesador** uC	**KoreBot** uP	**Gumstix** uP	**EyeBot** uC	**CMUcam3** uP
Procesador	Motorola 8bit 68HC11 2 MHz	Microchip 8bit PIC16C715 4MHz	BasicAtom-Pro16bit H8/3664F 100 K IPS	Intel 32 bit XSCALE PXA-255 400MHz	Verdexpro Marvell PXA270 400-600 MHz	Motorola 68332 32bit Controller 25 MHz	NXP LPC2106 ARM7 60 MHz
Potencia comp. estimada	0.5 - 1 MIPS	1 - 2 MIPS	0.1 MIPS	200 - 300 MIPS	200 - 500 MIPS	15 - 20 MIPS	40 - 50 MIPS
Software de programación	Interactive C	Cricket Logo	AtomPro Language	Linux 2.4.19	Linux 2.6.27	RoBIOS	Custom C code
Memoria	32 kB	4 kB	4 kB	64 MB	64 MB	1 MB	64 KB
Características y soporte para Senso-res/Actuadores	- 4 motores DC (1 A) - pantalla LCD - 7 entradas analógicas - 9 entradas digitales - 2 IR (trans/recep) - Bocina	- 4 motores DC - 1 sensores IR -6 sensores digitales - 8 servos - bocina	- Bocina -PlayStation Conector - Leds,botones - 16 pines E/S - 16 bus (Servos, motores DC, pines E/S) - Socket 24/28 pines	- 1KB-250 Interfaz (3 Seriales, 4 USB, 1 I2C, 1 SPI, 1 AC97, 53 GPIOs) - 2 RS232 - 1 USB Cliente	- USB host - 3 UARTs - hasta 90 GPIOs - 1 I2C Bus - 1 Bluetooth	- camera digital - 2 motores DC con encoders - 12 servos -6 sensores IR (o 6 entradas digitales) - 2 parachoques (o 2 entradas digitales) - 6 entradas analógicas - LCD, bocina	- sensor RGB color 352x288 - 4 servos
Protocolos	- SPI	- IR	- UART	- I2C - UART - SPI - USB	- I2C - UART - USB - Bluetooth	- UART	- UART - SPI
Expansión	- conector SPI - conector analógico - Tarjeta de Expansión (sensores analógicos, LEGO, salidas digitales, servos)	- Bus Port (tarjetas de display, motores DC, servos, reles, leds, etc)	- SSC-32 - SaberTooth	- KoreMotor (4 motores DC) - KoreConnect (RS232 y USB) - KoreSound (audio E/S) - KoreIO (pines E/S analógicos, salidas digitales, servos)	-RoboAudio -Robostix (SPI, I2C) -GPSstix (GPS,Audio,LCD,USB)	- EyeCam (sensor CMOS) 640x480 pixels	–
Precio (USD)	$ 299	$ 139	$ 25 + $60 processsor	$ 440 + $190 expan.	$159 + ($30 - $130)	$ 1,000	$ 239
Tamaño (cm)	10.8 x 8	5.7 x 4.8	10 x 8	8.5 x 5.7	8 x 2	10.6 x 10	5.5 x 5.7
Peso (grs)	200 aprox.	<200	90 aprox	35	8	115 aprox	200 aprox
Alimentación	12 v DC, 500 mA	5 - 6 v DC, 500 mA	9 - 5 v DC, 500 - 250 mA	5 v DC, 250 mA	3.6V - 5.0V DC, 500 mA aprox	7.2 - 9 v DC, 270 mA	5v DC, 130 mA

Tabla 2.1: Resumen de las propiedades de algunas de las plataformas comerciales más utilizadas para robots móviles que existen en el mercado. Datos tomados a la fecha de Diciembre-2008. Excepto por las propiedades de "peso" y "alimentación" tomadas a la fecha de Febrero-2009

Capítulo 3

Sistemas embebidos y FPGAs

En esta sección se da una breve introducción teórica a los conceptos de Sistemas Embebidos (SE) y de los FPGAs, ya que la la plataforma propuesta en este trabajo esta basada en realizar un SE en un FPGA.

3.1. Sistemas embebidos

Un sistema embebido o empotrado (*Embedded System*) es un sistema computacional de propósito especial que se integra en un solo circuito integrado (o pocos circuitos integrados íntimamente acoplados en un espacio reducido) un sistema computacional completo, esto es, integra procesadores, memorias RAM, ROM, periféricos, *buses*, etc. Un SE está diseñado para desempeñar una función/tarea en específico y usualmente con pocos requerimientos. Ya que un SE está enfocado para tareas específicas, éste se puede optimizar, reduciendo el costo y tamaño del producto final. Así que dos de las características principales son el precio y el consumo. Puesto que los sistemas embebidos se pueden fabricar por decenas de millares o por millones de unidades, una de las principales ventajas es que se pueden reducir los costos, ya que suelen usar un procesador y una memoria relativamente pequeños (Don et al., 2005).

Los sistemas embebidos (SE) son sistemas altamente integrados. Suelen ocupar muy poco espacio y tener un consumo de potencia reducido. Con la entrada al mercado de las FPGAs los SEs aún se pueden reducir más, y se puede hablar de sistemas SoC (*system on a chip*, o sistema en un circuito integrado), es decir todo un sistema compuesto de procesador, memoria y circuitos de apoyo personalizados pueden entrar en un solo circuito integrado. La importancia de un SE también radica en su potencia computacional, ya que está optimizado en cuanto a espacio y capacidad de procesamiento y sólo ocupa los recursos necesarios para desempeñar la tarea para el que fue hecho.

Un SE puede llegar a funcionar a una escala competitiva de MIPS (Mega Instrucciones Por Segundo), mientras que una PC puede llegar a funcionar a varias GOPS (Giga Operaciones Por Segundo), pero la desventaja de esto último es la necesidad de llevar una Laptop o una PC en el sistema, siendo para cierto tipo de dispositivos (e.g. como algún robot móvil pequeño) una carga grande y pesada, además de funcionar con procesadores estrictamente secuenciales, impidiendo así explotar el paralelismo necesario para algunas aplicaciones.

Físicamente, un SE puede ser desde dispositivos de uso diario como relojes digitales, reproductores MP3, celulares, lavadoras hasta instalaciones grandes como un controlador de luces de tráfico, controladores en fábricas o sistemas en plantas nucleares. En términos de complejidad un SE puede ir desde algo simple, por ejemplo con un microcontrolador hasta sistemas más elaborados con múltiples unidades, periféricos y redes montadas dentro de un mismo chasis o estructura física.

3.1.1. Características

Dentro de las características más importantes de un sistema embebido se tienen las siguientes:

- Un SE está diseñado para realizar una o algunas tareas específicas, a diferencia de una computadora de propósito general, diseñada para desempeñar múltiples tareas de diferente índole.
- Un SE no siempre es un bloque separado o completo por sí solo, muchas veces es introducido físicamente dentro del dispositivo al cual va a controlar. Por ejemplo en un automóvil se tienen decenas de sistemas embebidos funcionando, sin embargo descubrirlos a simple vista resulta difícil porque se encuentran en forma de aire acondicionado, reproductor de música, automatización de frenos, etc.
- El software escrito para un SE, a veces es llamado "*firmware*", es guardado en una memoria FLASH de los *chips*, más que en un disco duro. Este software a menudo se ejecuta y maneja pocos recursos hardware.

En general, un SE consiste de un sistema con microprocesador cuyo hardware y software están específicamente diseñados y optimizados para resolver un problema concreto eficientemente, como por ejemplo aparatos electrodomésticos, controladores de procesos en una empresa, controladores de semáforo, etc. Normalmente un SE interactúa continuamente con el entorno para vigilar o controlar algún proceso mediante una serie de sensores. El hardware de un SE se diseña normalmente a nivel de circuitos integrados, o de interconexión de PCBs, buscando la mínima circuitería y el menor tamaño para una aplicación en particular. Otra alternativa es el diseño a nivel de PCBs que consiste en el ensamblado de placas con microprocesadores comerciales que responden normalmente a un estándar como el PC-104. Esta última solución acelera el tiempo de diseño pero no optimiza ni el tamaño del sistema, ni el número de componentes utilizados, ni el coste unitario. En general, un sistema embebido simple contará con un microprocesador, memoria, unos cuantos periféricos de E/S y un programa dedicado a una aplicación concreta almacenado permanentemente en la memoria. El término embebido o empotrado hace referencia al hecho de que la "computadora" está encerrada o instalada dentro de un sistema mayor y su existencia como "computadora" puede no ser aparente.

Muchos SEs son sistemas de tiempo real. Un sistema de tiempo real debe responder, dentro de un intervalo restringido de tiempo, a eventos externos mediante la ejecución de la tarea asociada con cada evento. Los sistemas de tiempo real se pueden caracterizar como blandos[1] o duros (*soft / hard*). Si un sistema de tiempo real blando no cumple con sus restricciones de tiempo, simplemente se degrada el rendimiento del sistema, pero si el sistema es de tiempo real duro no cumple con sus restricciones de tiempo, el sistema fallará.

Un SE complejo puede utilizar un sistema operativo como apoyo para la ejecución de sus programas. La utilización de un sistema operativo de tiempo real (RTOS) es común en los sistemas embebidos, un RTOS es un sistema operativo diseñado y optimizado para manejar fuertes restricciones de tiempo (Li & Lao, 2003).

3.1.2. Componentes

Como muestra la figura 3.1 un sistema embebido está compuesto por cuatro elementos principales: CPU, memoria, *buses* y periféricos, aunque puede haber otros componentes dependiendo del sistema. El CPU es el elemento procesador de datos y puede ser un microprocesador, microcontrolador, DSP o bien un FPGA. En los elementos de memoria van almacenados tanto los datos como el propio programa que ejecuta el sistema, también la memoria puede servir para guardar un sistema operativo o bien como memoria cache. En cuanto a los periféricos es todo

[1] En un sistema de tiempo real blando se intentan cumplir los plazos de ejecución, pero en caso que alguna tarea finalice un poco tarde no sucede ningún desastre. Puede soportar que algunos tiempos no sean alcanzados.

aquello que el sistema tiene para comunicarse con el "mundo" exterior, pueden ser elementos de comunicación, sensores, motores, antenas, etc., aunque puede haber periféricos que funcionen internamente. Y finalmente debe haber conexiones que puedan comunicar y unir a todos estos elementos y para eso están los *buses*, los encargados de colocar las "reglas" de intercambio de datos entre los distintos componentes del sistema.

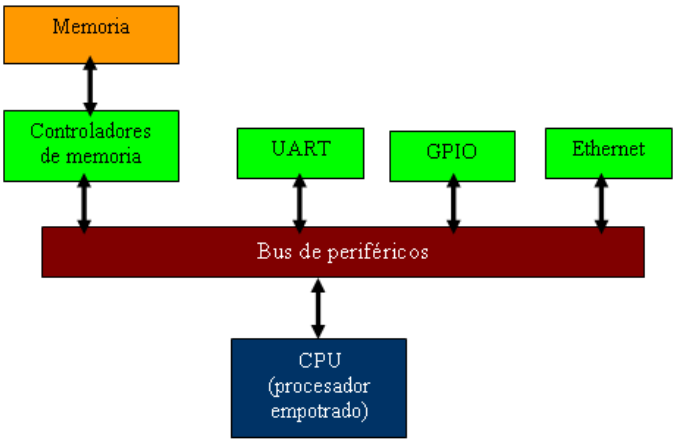


Figura 3.1: Componentes que forman un sistema embebido: Procesador, Buses, Memoria y Periféricos. Una de las características es que todo se encuentra en un solo circuito integrado o en un conjunto reducido de circuitos integrados

Hablando en más detalle de los componentes de un SE, se tiene que en la parte central se encuentra el microprocesador, microcontrolador, DSP, etc. Es decir la CPU o unidad que aporta el procesamiento de datos al sistema y la gestión de tareas y recursos hardware. La comunicación adquiere gran importancia en los SEs. Lo normal es que el sistema pueda comunicarse mediante interfaces estándar de cable o inalámbricas, por ejemplo RS232, RS485, SPI, I^2C, CAN, USB, IP, WiFi, GSM, GPRS, etc. Puede haber también un subsistema de presentación que suele ser una pantalla gráfica, táctil, LCD, alfanumérico, etc.

Se denominan actuadores (también conocidos como activadores o accionadores) a los posibles elementos electrónicos que el sistema se encarga de controlar. Puede ser un motor eléctrico, un conmutador tipo rele, unos leds, etc.

También puede haber un módulo de reloj que es el encargado de generar las diferentes señales de reloj a partir de un único oscilador principal. El tipo de oscilador es importante por varios aspectos: por la frecuencia necesaria, por la estabilidad y por el consumo de corriente requerido. Otro módulo importante es el de energía, el cual se encarga de generar las tensiones y corrientes necesarias para alimentar los diferentes circuitos del sistema. El consumo de energía puede ser determinante en el desarrollo de algunos SEs que necesariamente se alimentan con baterías, por lo que el tiempo de uso del SE suele ser la duración de la carga de las baterías.

3.2. FPGAs

Los FPGAs (*Field Programmable Gate Arrays*) están revolucionando la manera en que los diseñadores de sistemas implementan lógica digital, reducen radicalmente los costos y el tiempo de desarrollo para implementar sistemas digitales completos dentro de un solo *chip* (Xilinx, Diciembre 2008).

Un diseñador de sistemas electrónicos dispone de diversas opciones para implementar lógica digital, incluyendo dispositivos lógicos discretos, frecuentemente llamados Circuitos Integrados

de pequeña escala (SSI); dispositivos programables tales como Arreglos de Lógica Programables (PALs o PLDs); y Arreglos de Compuertas Programables en el Campo(*FPGAs - Field Programmable Gate Arrays*); o bien Circuitos Integrados de Aplicación Específica (ASIC). Los diseñadores que usan FPGAs pueden definir funciones lógicas en un circuito y revisar estas funciones como sea necesario una y otra vez. Así, los FPGAs pueden diseñarse y verificarse en poco tiempo, a diferencia de las otras técnicas.

Un FPGA (ver figura 3.2) es un dispositivo que contiene bloques de lógica cuya interconexión y funcionalidad se puede programar. La lógica programable puede reproducir desde funciones tan sencillas como las llevadas a cabo por compuertas lógicas o un sistema combinacional, hasta complejos sistemas en un chip (SoC). Los FPGAs son dispositivos programables de propósito general con gran capacidad de integración. Éstos son matrices de compuertas programables en el campo, es decir, la funcionalidad del FPGA es definida por un programa de usuario en lugar de definirla el fabricante. La enorme libertad disponible en la interconexión de dichos bloques otorga a los FPGAs una gran flexibilidad.

Figura 3.2: FPGAs de Xilinx y Altera. Figuras tomadas de Xilinx (Diciembre 2008)

Los FPGAs se utilizan en aplicaciones similares a los ASICs, sin embargo, son más lentos y tienen un mayor consumo de potencia, pero tienen un menor consumo comparado con los procesadores de las PCs. A pesar de esto, los FPGAs tienen las ventajas de ser reprogramables (lo que añade una enorme flexibilidad al flujo de diseño), permiten implementar prácticamente cualquier tipo de lógica, permiten el paralelismo, entre otras. Existen dos principales compañías dedicadas a la fabricación de estos dispositivos así como de sus herramientas: Xilinx y Altera.

Una tendencia reciente ha sido combinar los bloques lógicos e interconexiones de los FPGA con microprocesadores y periféricos relacionados para formar un SoC. Ejemplos de tales tecnologías híbridas pueden ser encontrados en los dispositivos Virtex-II PRO, Virtex-4 y Virtex-5 de Xilinx, los cuales incluyen uno o más procesadores PowerPC embebidos junto con la lógica del FPGA. Otra alternativa es hacer uso de núcleos de procesadores implementados con la lógica del FPGA. Esos núcleos incluyen los procesadores MicroBlaze y PicoBlaze de Xilinx, Nios y Nios II de Altera, y los procesadores de código abierto LatticeMicro32 y LatticeMicro8.

Muchos FPGA modernos dan soporte a la reconfiguración parcial del sistema, permitiendo que una parte del diseño sea reprogramada, mientras las demás partes siguen funcionando, todo esto mientras el sistema está funcionando. Este es el principio de la idea de la computación reconfigurable, o de los sistemas reconfigurables.

Hoy las capacidades en los FPGAs han evolucionado más allá de las capacidades básicas que poseían sus predecesores, y han incorporado bloques "*hard*" (tipo ASIC) de uso común, como memorias RAM, controladores de reloj y bloques DSPs entre otros, a continuación se describen los componentes básicos en un FPGA (ver figura 3.3):

- ***Bloques lógicos configurables (Configurable Logic Block - CLBs)***. El CLB es la unidad básica en un FPGA. Cada uno de éstos es una matriz configurable altamente flexible y puede ser configurada para soportar lógica combinacional, lógica secuencial, registros de desplazamiento, o simplemente memoria RAM.

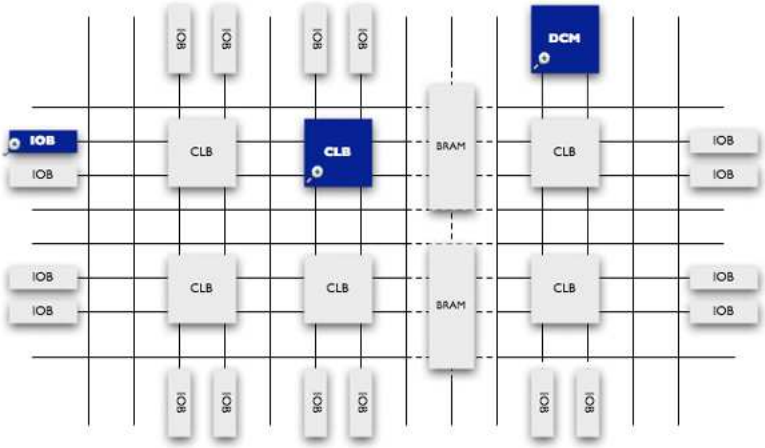


Figura 3.3: Estructura interna general en un FPGA. Aquí se observan los componentes principales en un FPGA: CLBs, IOBs, BRAM, DCMs y una matriz de interconexión. Figura tomada de Xilinx (Diciembre 2008)

- ***Recursos de Interconexión.*** Mientras que el CLB proporciona la capacidad lógica, una serie de rutas flexibles de interconexión permiten conectar CLBs con CLBs o bien con puertos E/S del FPGA, o bien algún otro recurso del FPGA. El software es el encargado de realizar el archivo para las tareas de enrutamiento o bien el usuario puede personalizar las conexiones, lo cual reduce notablemente el tiempo de diseño.

- ***Bloques Entrada-Salida (IOBs)***. Hoy los FPGAs proporcionan centenas de pines de E/S y docenas de estándares I/O proporcionando así una buena interfaz hacia otros sistemas. Los bloques en un FPGA están formados por bancos en donde cada uno de ellos soporta varios estándares I/O.

- ***Memoria RAM embebida***. La memoria RAM es muy común en cualquier sistema digital, así que este tipo de memoria está disponible en la mayoría de los FPGAs. Esto provee la característica de que el dispositivo FPGA tenga memoria dentro del *chip* y así optimizar los accesos a memoria.

- ***Controladores de reloj (Digital clock managers - DCMs)***. Estos controladores de reloj están disponibles en todos los FPGAs de Xilinx, los cuales proporcionan recursos para una buena calidad en la señal de reloj, se pueden realizar divisores de frecuencia, de fase, entre otras cosas.

El diseñador cuenta con la ayuda de entornos de desarrollo especializados en el diseño de sistemas a implementarse en un FPGA. Un diseño puede ser capturado ya sea como esquemático, o haciendo uso de lenguajes de programación especiales conocidos como HDLs (lenguajes de descripción de hardware). Los HDLs más utilizados son: VHDL, Verilog y ABEL. Existen otros entornos de programación de FPGAs así como herramientas especializadas para tratar con sistemas complejos. Como ejemplo se tiene la herramienta EDK, que es un conjunto de aplicaciones para el desarrollo de SEs, otro ejemplo es PlanAhead que sirve para optimizar los diseños y realizar cómputo reconfigurable.

Las aplicaciones de los FPGAS incluyen a los DSP (procesamiento digital de señales), sistemas de radiofrecuencia, sistemas aeroespaciales y de defensa, prototipos de ASICs, sistemas de imágenes para medicina, sistemas de visión para computadoras, reconocimiento de voz, bioinformática, *cores* para robótica, entre otras. Cabe mencionar que su uso en otras áreas es cada

vez mayor, sobre todo en aquellas aplicaciones que requieren un alto grado de paralelismo, como en aplicaciones de visión por computadora.

Con estas propiedades de los FPGAs, resulta que se puede integrar un Sistema Embebido y lógica de periféricos y co-procesadores en el mismo chip, como se ve a continuación.

3.3. Sistemas embebidos en FPGAs

En el pasado, los sistemas digitales eran construidos con muchas partes discretas, incluyendo por ejemplo FPGAs, DSPs, Microprocesadores, memoria, dispositivos de E/S, etc. Mientras que las herramientas de diseño lógico fueron y siguen siendo usadas para especificar la funcionalidad individual de cada parte y también para integrar todas estas partes en un solo sistema.

Por el lado del CPU, el cual a menudo es una parte discreta. La interfaz del CPU con la lógica, memoria y puertos E/S es desarrollada mediante herramientas software de los fabricantes. La parte software del sistema es también implementada con herramientas especializadas para código de alto nivel y desarrollo de sistemas operativos. La interfaz hardware-software es realizada también con estas herramientas.

En los últimos años de la década de los 90s, muchas de las funciones lógicas fueron integradas dentro de un mismo dispositivo. En cuanto a la memoria esta todavía se encontraba fuera del *chip* en donde se encontraba el sistema embebido, pero conforme avanzó el tiempo se desarrollaron dispositivos que contenían una buena cantidad de memoria disponible (unas cuantas centenas de kilo bytes). Esto permitía que cada vez más lógica fuera implementada y desarrollada como una unidad simple, más que en partes discretas donde se requerían interfaces adicionales. Sin embargo, el CPU seguía siendo una entidad aparte, y lo sigue siendo en ciertos tipos de sistemas embebidos.

Hoy en día, con el avance de tecnología en las plataformas FPGA y SoC ASICs, se han fusionado en un mismo *chip* varias de las funcionalidades lógicas, y en algunos casos todas las que se necesitan para un sistema. Ahora los procesadores vienen incluidos en el *chip* (e.g. FPGAs). Para el diseño de sistemas programables hardware y software, ambas partes requieren de la unión de silicio, de *cores* (IP) tanto a nivel hardware como a nivel software y de todo un conjunto de herramientas que funcionen como unidad (Xilinx-3, Diciembre 2008), tal y como se observa en la figura 3.4.

El diseño de un sistema digital típico involucra una gran cantidad de circuitería lógica hecha a la medida, aunque también involucra componentes ya pre-fabricados como procesadores, controladores de memoria y varios tipos de interfaces de E/S. En un sentido tradicional para el diseño de tales sistemas, una nueva tarjeta de circuito integrado (CI) es creada para el sistema completo hecho a las necesidades del usuario, y esto a menudo es una tarea ardua y tediosa, sobre todo diseñar la circuitería hecha por el usuario.

Los FPGAs son una buena opción para implementar sistemas digitales o SEs ya que:

- Ofrecen una capacidad lógica de gran tamaño, llegando hasta varios millones de compuertas equivalentes e incluyendo recursos de memoria integrada en el chip.
- El FPGA es un gran "pizarrón en blanco" de lógica en donde se puede implementar cualquier subsistema digital. Desde lógica de unión (*glue-logic*) que tradicionalmente se implementaban con dispositivos TTL/CMOS en los 80's, hasta módulos procesadores digitales. Un FPGA puede contener de 10,000 hasta varios millones de compuertas lógicas equivalentes en un solo *chip*.
- Los FPGAs incluyen *cores* o módulos hardware especializados, tales como módulos DSP, multiplicadores embebidos, PLLs, controladores de periféricos como VGA, RS-232, I2C, USB, etc.
- Varios módulos implementados en un FPGA pueden funcionar al mismo tiempo, logrando con esto un alto grado de paralelismo necesario y útil en varias aplicaciones.

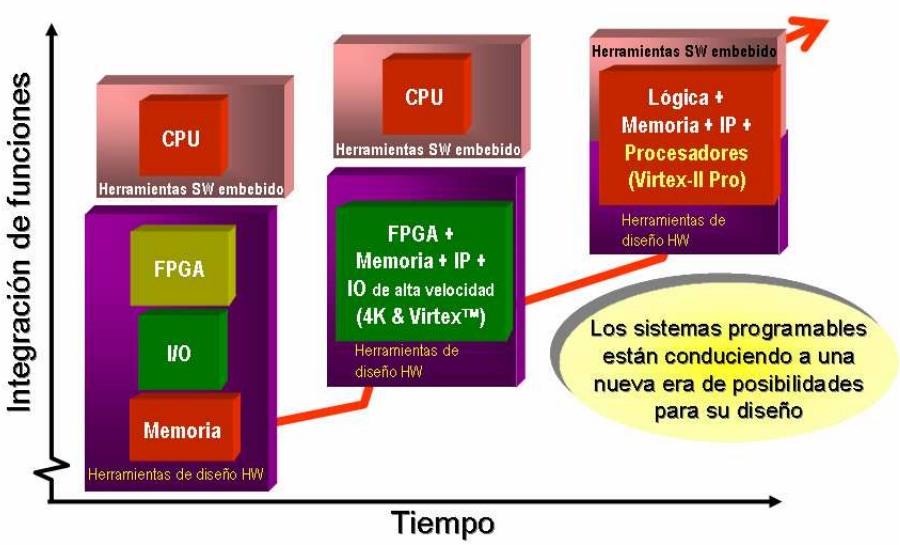


Figura 3.4: Niveles de integración en un diseño digital a lo largo de los años. Primero se tenían entidades hardware totalmente separadas, después algunas de estas entidades fueron integradas en un solo dispositivo. Ahora se pueden tener sistemas completos en el mismo dispositivo, esto es, sistemas que integran lógica digital, memoria, *cores* y procesadores. Figura tomada de Xilinx-3 (Diciembre 2008)

- Los FPGAs incluyen una gran variedad de interconexiones estándar como DDR-SDRAM, PCI, protocolos seriales de alta velocidad, etc. También dan soporte a varios estándares en cuanto a voltajes en sus pines como LVTTL, CMOS, LVCMOS, etc.
- La funcionalidad final del FPGA es definida por el usuario y es reprogramable una y otra vez. Esto evita la necesidad de largos procesos de manufactura y costos para el diseño del sistema.

En el diseño de SEs con FPGAs, puede haber mucha flexibilidad en cuanto a decidir qué partes del sistema se van al FPGA y que partes deberán permanecer en chips separados. Posiblemente lo mejor sería que todo el sistema estuviera en el FPGA, es decir que se pueda tener un SoC (*System on Chip*), incluyendo el procesador, los buses, sus periféricos y memoria. Esto ya es posible gracias al gran avance que se han realizado en estos dispositivos y cada día soportan mayor capacidad de lógica.

Hablando de los procesadores implementados en FPGAs, existen dos tipos: *hard processors y soft processors*. Un procesador tipo *hard* es un circuito pre-diseñado y pre-fabricado que es "insertado" tal cual dentro de un FPGA. Una alternativa más flexible es un procesador tipo *soft*, el cual existe como un código escrito en un lenguaje de descripción hardware, como VHDL y es implementado con los recursos del FPGA, usando la lógica de CLBs, LUTs, recursos de memoria, etc. En la figura 3.5 se observan las tres soluciones que ofrece la empresa Xilinx para procesadores. Adicionalmente a la potencia computacional de estos procesadores, hay que sumar la potencia computacional lograda con la ayuda de co-procesadores que se pueden implementar en paralelo en el mismo FPGA, logrando alcanzar desempeños de varios GOPS (giga operaciones por segundo) en un solo dispositivo, comparable o en algunos casos superior a lo que da una PC convencional. Además de las ventajas de tamaño, costo y consumo de potencia que se logran.

En cuanto a las herramientas para el desarrollo de SEs, éstas deben considerar dos aspectos principales: 1) la creación del sistema hardware, y 2) el desarrollo del software que corre en el procesador. Los fabricantes de FPGAs proporcionan herramientas automáticas que facilitan

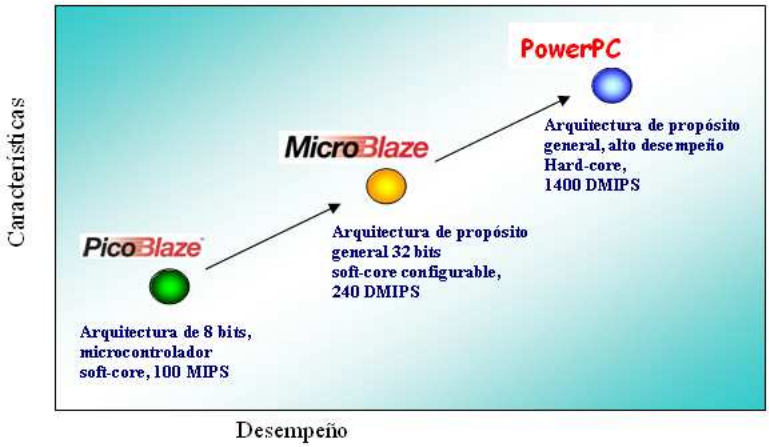


Figura 3.5: Soluciones en procesadores de Xilinx: Picoblaze©, Microblaze©y PowerPC©son los tres procesadores que pueden ser usados junto con FPGAs. Figura basada en Xilinx-3 (Diciembre 2008)

ambas partes del diseño, una de estas herramientas y la que se usa en este trabajo es el *Embedded Development Kit* (EDK) de Xilinx. Para la creación de la parte hardware, estas herramientas deben permitir mediante un entorno amigable el construir el sistema haciendo uso de bloques predefinidos para el propio procesador, controladores de memoria, *buses*, circuitos DSPs, y varios módulos de comunicación; para este fin EDK tiene *Xilinx Platform Studio* (XPS). Por lo que respecta a la parte software, éste debe permitir el control de estos componentes hardware de la plataforma, sobre todo los periféricos, también la parte software es la parte "inteligente" del sistema, es la aplicación en donde el programador le dice al sistema qué hacer dependiendo de las entradas que reciba el sistema. Las herramientas que permiten el desarrollo de la parte software generalmente vienen con editores de programas, depuradores, compiladores y soporte para implementar un tipo de sistema operativo. Para este caso EDK tiene una herramienta llamada *Software Development Kit* (SDK) que permite realizar pequeños programas y depurarlos. Pero también existe una gran variedad de fabricantes que desarrollan estas herramientas software para distintos tipos de plataformas (*third party*), como lo son Petalogix, LinuxWorks, Mentor Graphics, Wind River Systems, etc. (Xilinx-3, Diciembre 2008).

3.4. Resumen del capítulo

En el presente capítulo se realizó una revisión de lo que son los sistemas embebidos y los FPGAs, ahora se tiene un panorama general de lo que son este tipo de dispositivos, lo que pueden llegar a hacer y de las ventajas que se pueden lograr utilizándolos en sistemas embebidos. Los dispositivos FPGA son una solución atractiva para las limitantes que presentan las plataformas comerciales. Como se mencionó al final del capítulo 2, una opción interesante sería tener una plataforma que sirva para diferentes usos y objetivos, aparte de ofrecer recursos suficientes para tener una potencia computacional aceptable (como la KoreBot o Gumstix). Los FPGAs pueden cubrir estas limitantes por sus capacidades de reconfiguración (cambiar el hardware de la plataforma usando el mismo dispositivo), paralelismo (acelerar tareas mediante módulos hardware que trabajan al mismo tiempo) y flexibilidad (realizar plataformas hechas a la medida dependiendo del proyecto). En el siguiente capítulo se presenta la plataforma FPGA propuesta y se detalla su arquitectura hardware/software.

Capítulo 4

Plataforma FPGA propuesta

4.1. Introducción

En este capítulo se presenta el diseño de la plataforma FPGA propuesta para este trabajo. Ya que la plataforma FPGA es un sistema empotrado, ésta se encuentra dividida en su parte hardware y su parte software, las cuales se llamarán arquitectura hardware y arquitectura software, respectivamente.

Arquitectura Hardware. Es toda la infraestructura hardware necesaria para dar soporte a los elementos del robot. En específico, esta plataforma da soporte a un robot móvil que se comunique a bajo nivel por medio de un protocolo serie. También la plataforma da soporte a otros aditamentos que sirven parra mejorar la funcionalidad del robot, estos aditamentos pueden ser: brazo robot, sonares, sensores infrarrojos, brújula, co-procesadores, etc.
La infraestructura hardware consta básicamente de cuatro elementos: procesador, memoria, *buses* y periféricos. Todo implementado dentro del FPGA, con esto se ahorra espacio y circuitería extra. La finalidad de la plataforma FPGA hardware es ofrecer los recursos necesarios para soportar un robot móvil que tenga una forma de comunicación a bajo nivel (generalmente serial) y que ofrezca una forma sencilla de incorporar elementos extras al robot, particularmente sensores y actuadores comúnmente usados en robots móviles. Lo más importante es aprovechar las ventajas que ofrecen este tipo de dispositivos para mejorar el desempeño del robot, como flexibilidad, paralelismo y reconfiguración.

Arquitectura Software. Es toda la infraestructura software necesaria para controlar a bajo y a mediano nivel tanto al robot como a sus elementos extras. También esta plataforma tiene un sistema operativo (SO) basado en Linux sobre el cual se encuentran funcionando todas las bibliotecas que controlan a cada uno de los elementos del robot de prueba.
Las funciones que se ejecutan en el SO tienen la característica de estar escritas en un lenguaje de alto nivel además de ser fáciles de usar para un programador de robots convencional. La finalidad de esta plataforma es ofrecer funciones para el control del robot y de sus elementos extras así como el de dejar abierta la posibilidad de incrementar la arquitectura software para dar soporte a más Bibliotecas, estructuras de datos, capas de abstracción, etc.

En la figura 4.1 se muestra la propuesta de una tarjeta FPGA genérica, es decir, una tarjeta que pueda servir para diferentes plataformas robóticas, la idea de diseño de esta tarjeta es que el FPGA pueda ser configurado por medio de archivos bistream y así poder implementar sistemas embebidos para diversas aplicaciones. Esta tarjeta fue pensada para albergar una buena cantidad de lógica dentro del FPGA y con algunos puertos de comunicación útiles y sobretodo una buena cantidad de pines E/S que darán soporte a sensores, actuadores, cámaras, etc. que se usen en

aplicaciones robóticas. Aunque la tarjeta usada para este trabajo es una que vende la compañía Xilinx comercialmente, la fabricación de la tarjeta de la figura 4.1 es bastante viable.

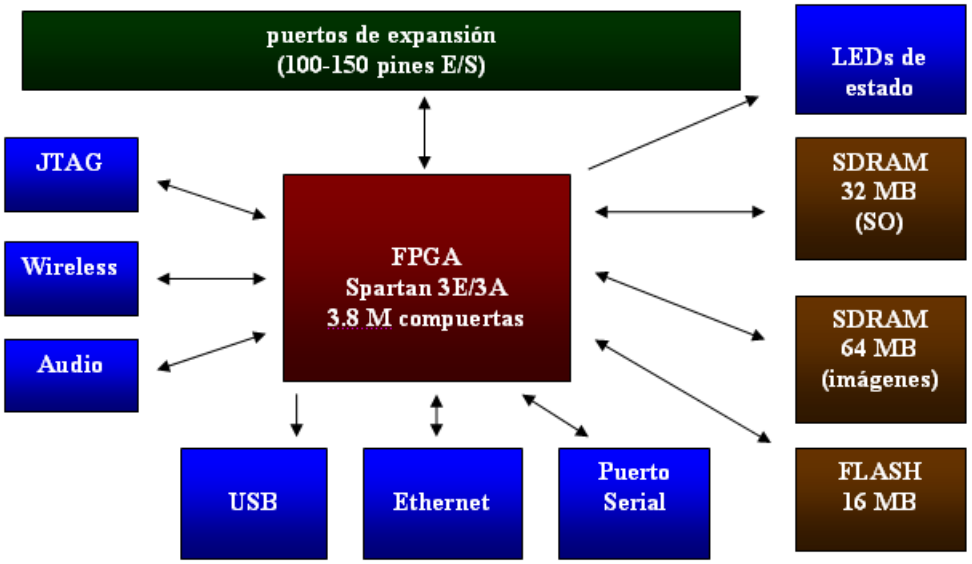


Figura 4.1: Tarjeta FPGA propuesta para probar diferentes platafomas robóticas, en donde el FPGA se configura con diferentes archivos *bitstream*. La plataforma FPGA realizada en este trabajo fue implementada en una tarjeta comercial de la empresa Xilinx

En las secciones siguientes se describe en detalle el diseño tanto de la parte hardware como de la parte software de la plataforma FPGA, así como la funcionalidad de la misma, es decir, cual es la interfaz al usuario, forma de programación, dispositivos que se tiene soporte, etc. Finalmente se hace una introducción a la implementación de la plataforma FPGA sobre una tarjeta de desarrollo.

4.2. Arquitectura hardware

La arquitectura hardware, como su nombre sugiere, consiste en el sistema empotrado (procesador, memoria, *buses* y periféricos) que se encuentra implementado dentro del FPGA, esta arquitectura sirve para comunicarse con el robot móvil y sus aditamentos (e.g. brazo robot, sonares, brújula, etc.). Además, esta arquitectura tiene otros módulos hardware que de alguna forma ayudan a mejorar el desempeño del robot, por ejemplo, el co-procesador para imágenes que aprovecha el paralelismo del FPGA para mejorar las tareas de convolución.

En la figura 4.2 se observa la arquitectura hardware propuesta. Esta arquitectura tiene la finalidad de ofrecer el soporte para controlar robots móviles con interfaz serial, ya que gran parte de los robots móviles comerciales tienen una forma de interfaz serial para comunicarse con el robot a bajo nivel. Más aún si el robot no se comunica de forma serial, el protocolo de comunicación a bajo nivel del robot puede ser implementado con la lógica del FPGA. También la plataforma a través de un puerto de expansión puede dar soporte a otros aditamentos comunes en Robótica, como lo son sonares, brújula, sensores infrarrojos, brazos manipuladores, etc. Nuevamente la forma de comunicación de estos aditamentos puede ser implementada con la lógica del FPGA, así que en este sentido se puede soportar la gran mayoría de los sensores y actuadores que se venden comercialmente para aplicaciones de robótica.

En general la arquitectura hardware de la figura 4.2 está compuesta por los cuatro componentes principales de un sistema empotrado: procesador, memoria, *buses* y periféricos. Existe

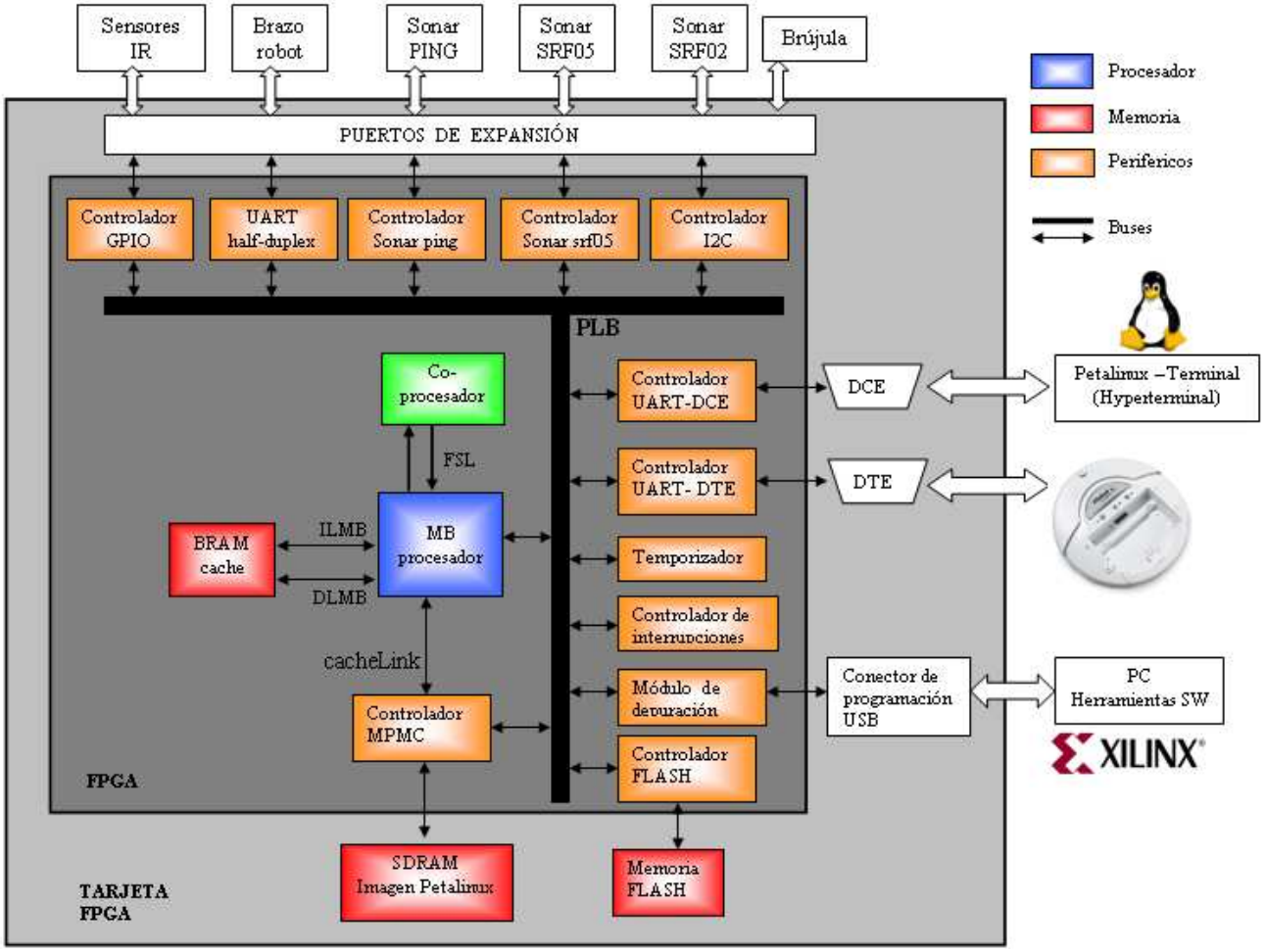


Figura 4.2: Parte hardware de la plataforma FPGA. Arquitectura hardware propuesta para dar soporte al robot móvil y sus aditamentos, en donde se observa la interconexión entre el procesador, la memoria y sus periféricos. Esta arquitectura fue realizada sobre una tarjeta FPGA de desarrollo en específico, pero se puede construir una tarjeta FPGA propia sólo con los elementos necesarios y así poder disminuir el tamaño

un quinto elemento añadido al sistema que es el 'co-procesador', del cual se habla en breve. En la figura 4.2 se observa el procesador como elemento central, el cual interactúa con tres tipos de memoria: BRAM, SDRAM y FLASH. La memoria BRAM sirve para implementar memoria cache, la memoria SDRAM guarda la imagen del SO así como aplicaciones del usuario y la memoria FLASH sirve para realizar almacenamiento permanente del SO. En la misma figura se tienen también los periféricos del sistema los cuales son básicamente funcionan como controladores para interactuar con los dispositivos hardware externos (e.g. robot móvil, sensores, actuadores, PC, etc.), aunque se tienen algunos periféricos internos como el temporizador y el controlador de interrupciones, los cuales no tienen comunicación con el exterior del sistema. Se tiene además un co-procesador, el cual interactúa directamente con el procesador con la finalidad de acelerar ciertas tareas aprovechando el hardware paralelo que puede ser implementado en los FPGAs. Finalmente están los *buses* necesarios para interconectar a todos los elementos del sistema.

A continuación se describe la funcionalidad de cada uno de los cinco componentes del sistema.

4.2.1. Procesador

El procesador es el cerebro del sistema que se encarga del procesamiento de los datos así como de ejecutar las aplicaciones software que estén cargadas en memoria. El tipo de procesador usado para este trabajo es un *soft-processor* Microblaze (Xilinx-4, Diciembre 2007), el cual se implementa con los recursos básicos del FPGA (*Slices, LUTs, FlipFlops,* etc.).

El usar este tipo de procesador permite tener una arquitectura flexible, manteniendo el balance entre buen desempeño y tamaño de implementación. Este tipo de procesador permite cumplir con los objetivos fundamentales que requiere cualquier proyecto de sistemas embebidos. Dentro de estos objetivos se tiene en primer lugar la *"reducción del costo total del producto"*, esto es, al integrar uno o mas procesadores Microblaze y que interactúen con diversos periféricos, todo esto dentro del mismo FPGA, se puede reducir el número de componentes a usar e implementar y se reduce también la complejidad del diseño de la tarjeta, disminuyendo así el costo total del producto. Otro objetivo requerido normalmente es *"evitar que el diseño se vuelva obsoleto"*, esto es, usar un procesador suave *(soft-processor)* permite fácilmente migrar a FPGAs que tengan arquitecturas compatibles y para la mayoría de las actualizaciones de las herramientas software del fabricante. Otro de los objetivos es el "alto nivel de programación" que se puede lograr, esto es, con las soluciones *third party*[1] que ofrecen otros fabricantes se tiene un buen abanico de herramientas software (como Linux embebido) para crear una gran variedad de aplicaciones que cumplan con los requerimientos que demanda el diseño. Y por último se pueden "alcanzar el desempeño requerido" para este tipo de sistemas, ya que se puede hacer una arquitectura hecha "a la medida" y con las ventajas que ofrecen los FPGAs, es decir, se pueden realizar co-procesadores a nivel hardware que de alguna manera aceleren el procesamiento de datos y así mejoren el desempeño del sistema, más aún se pueden añadir o eliminar tantos módulos hardware como se requieran, paralelizar procesos, etc. y todo usando el mismo dispositivo (Xilinx, Diciembre 2008).

4.2.2. Memoria

La memoria es una parte fundamental para la plataforma FPGA y en general para cualquier tipo de sistema empotrado. En este caso se tienen tres diferentes tipos de memoria: BRAM, DDR SDRAM y FLASH.

La memoria BRAM (*Block RAM*) se utiliza para implementar memoria cache en el sistema y de esta forma acelerar la transferencia de datos, la memoria BRAM son bloques de memoria implementados dentro del FPGA por lo que la comunicación tanto de datos como de instrucciones con el procesador es más rápida, ya que ambos están implementados dentro del mismo dispositivo.

La memoria DDR SDRAM es una memoria que se encuentra físicamente sobre la tarjeta de desarrollo y que está conectada al FPGA. La ventaja es que su capacidad de almacenamiento es mucho mayor que las memorias BRAM. La memoria DDR SDRAM se utiliza para almacenar la imagen del SO Linux que correrá sobre el FPGA. Esta imagen también contiene las aplicaciones del usuario, es decir, los programas del robot que el usuario programe para cierta aplicación. Esta memoria también puede ser utilizada para almacenar cantidades de datos importantes, como por ejemplo una imagen. Dependiendo de la finalidad y los recursos del robot, el tamaño de la memoria puede variar, un tamaño aceptable para la mayoría de las aplicaciones de este tipo de memorias es de 64 MB.

La memoria FLASH también es un tipo de memoria que se encuentra sobre la tarjeta de desarrollo y que también está conectada al FPGA, es un *chip* externo al FPGA, como la DDR SDRAM. La memoria FLASH se utiliza para el almacenamiento permanente de datos, en este caso se utiliza para almacenar tres archivos: el archivo *bitstream* (*.bit) necesario para programar el hardware del FPGA, el archivo de la imagen del SO Linux y un archivo *bootloader* que es

[1] *Third-party* es un término con el cual se conocen a empresas que desarrollan bibliotecas software que pueden integrarse como elementos de construcción adicionales. El término *third-party* no se refiere ni al fabricante ni al usuario, sino a un tercero.

un programa en C que se encarga de transferir la imagen del SO Linux hacia la memoria DDR SDRAM. La finalidad de la memoria FLASH es evitar la necesidad de descargar a través de la PC una y otra vez tanto el archivo *bitstream* como la imagen del SO, y de esta manera tener un mecanismo de arranque automático desde el momento en que se prende la tarjeta FPGA, sin la necesidad de tener una PC que programe la tarjeta.

4.2.3. Buses

Los *buses* son el medio por el cual se interconectan todos los elementos del sistema, los buses se encargan de proporcionar reglas para uniformar la transferencia de datos entre el sistema. La plataforma FPGA cuenta con cuatro tipos de *buses*: PLB (*Processor Local Bus*), LMB (*Local Memory Bus*), XCL (*Xilinx Cache Link*) y FSL (*Fast Simple Link*). Cada uno de ellos cuenta con características especiales que se adecuan dependiendo de su uso dentro del sistema. El *bus* PLB se usa para conectar la mayoría de los periféricos al procesador, el bus LMB se utiliza para comunicarse con la memoria BRAM, el *bus* XCL se utiliza para comunicarse directamente con la memoria DDR SDRAM y así poder implementar memoria cache al sistema, finalmente el *bus* FSL sirve para establecer una comunicación directa entre los co-procesadores implementados y el procesador. De esta forma, la finalidad principal de estos *buses* es interconectar de la mejor manera posible a los elementos del sistema dependiendo de lo que se quiera lograr. En la referencia (Xilinx-4, Diciembre 2007) se habla con más detalle de las características técnicas de cada uno de estos *buses*.

4.2.4. Periféricos

Un periférico es cualquier módulo hardware (*core*) dentro del sistema (e.g. procesador, controladores, lógica de usuario, temporizadores, etc.), para este trabajo los periféricos que no se consideraron como tales por tener un funcionamiento especial son el procesador Microblaze y el co-procesador de imágenes, éstos dos periféricos son tratados aparte. En este sentido en la plataforma FPGA propuesta, los periféricos se pueden dividir en tres categorías: controladores, lógica de usuario y módulos internos.

Los controladores (*drivers*) sirven para comunicarse con dispositivos hardware externos (e.g. sensores, actuadores, memorias, etc.), establecen un protocolo de comunicación con el dispositivo con que se quieren comunicar y de esta forma el procesador puede mandar/recibir datos desde/hacia el dispositivo hardware externo. Una característica importante de los controladores en FPGAs es que se pueden implementar una gran variedad de protocolos de comunicación (e.g. serial, SPI, I2C, CAN, etc.), incluso protocolos personalizados y hechos a la medida de las necesidades del proyecto, esto proporciona una gran flexibilidad al sistema. Un ejemplo ilustrativo de la función que realizan los controladores es el siguiente: la plataforma FPGA necesita comunicarse con algún robot móvil que tenga una forma de comunicación serial para controlarlo, así que se necesitaría implementar un módulo UART de comunicación serial para transmitir comandos de movimiento al robot así como recibir información de los sensores. El módulo UART es precisamente un ejemplo de un controlador. Por otro lado, varios microcontroladores disponibles en el mercado no poseen el conjunto de protocolos de comunicación requeridos para un sistema dado, usando dispositivos FPGA esto no es problema, ya que estos protocolos pueden ser implementados usando la lógica del FPGA.

La lógica de usuario son módulos hardware que implementan alguna función específica requerida por el sistema, generalmente son usados para desempeñar funciones que no se quieren hacer en software sino en hardware, ya por la rapidez y flexibilidad que se puede lograr, o bien debido al paralelismo y a la reconfiguración que poseen los dispositivos FPGA.

Los módulos internos son aquellos que no tienen una interfaz externa al FPGA, es decir, funcionan solamente dentro del sistema. Un ejemplo pueden ser los temporizadores (*timers*) los cuales generalmente proporcionan mediciones de tiempo hacia el procesador. Otro ejemplo es

el controlador de interrupciones, el cual gestiona las interrupciones que generan los diferentes periféricos del sistema y avisa al procesador que periférico es el que ocasionó la interrupción. Estos módulos internos no necesitan comunicarse con dispositivos externos. En (Xilinx-3, Diciembre 2008) se habla más en detalle de los *cores* que soportan los sistemas implementados en FPGAs de Xilinx.

4.2.5. Co-procesadores

Una de las ventajas de los procesadores *soft-cores* como Microblaze es su flexibilidad, es decir, el procesador utiliza solamente las características necesarias para una aplicación en específico. Otra de las principales ventajas es la habilidad de poder integrar *cores* IP hechos a la medida por el usuario (co-procesadores), los cuales pueden resultar en una considerable aceleración en el tiempo de ejecución en software, esto debido a que el algoritmo esta siendo ejecutado en paralelo en el hardware y no secuencialmente en el software. La figura 4.3 muestra un ejemplo ilustrativo de las ventajas de la ejecución hardware en paralelo, la rutina software necesita 12 ciclos de reloj para calcular el resultado G; mientras que la ejecución hardware necesita solamente 2 ciclos de reloj para calcular el mismo resultado, es un ejemplo ilustrativo de las ventajas que se tienen al usar un co-procesador para acelerar cálculos.

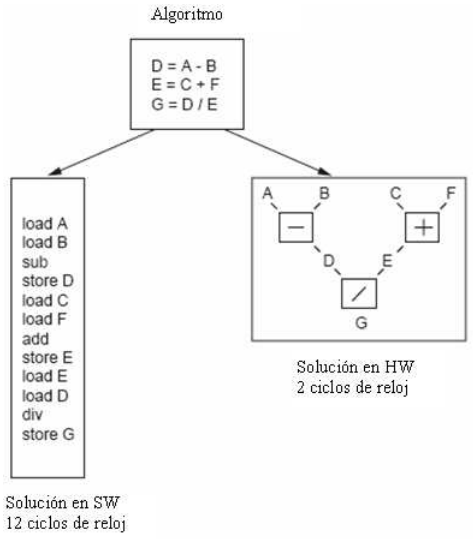


Figura 4.3: Ejemplo de un programa ejecutado en software y ejecutado en hardware, donde se observan las ventajas de la implementación hardware. Figura tomada de Xapp-Xilinx (2004)

Para la plataforma FPGA se propone realizar un co-procesador para tareas de procesamiento de imágenes, el cual desempeñe tareas de convolución aprovechando el paralelismo de los módulos hardware, pero lo importante es que esta técnica de utilizar co-procesadores puede extenderse a una buena variedad de aplicaciones para mejorar el desempeño de la plataforma robótica (e.g. algoritmos de visión, operaciones aritméticas complejas, procesamiento intensivo de datos, etc.).

En concreto la plataforma hardware propuesta cumple con dos aspectos importantes. El primero de ellos es ofrecer una arquitectura con los recursos suficientes para soportar a un robot móvil y elementos extras, tal y como lo haría cualquier plataforma disponible en el mercado (e.g. Gumstix, KoreBot). El segundo aspecto y más importante, es ofrecer ventajas que las plataformas disponibles en el mercado y que no están basadas en FPGA no ofrecen, es decir, la flexibilidad de poder implementar módulos hardware hechos a la medida para ayudar a realizar

ciertas tareas o acelerar algoritmos aprovechando el paralelismo. La principal ventaja de usar dispositivos FPGA, es que el FPGA es un "pizarrón en blanco" en donde se puede implementar hardware hecho a las necesidades del proyecto, además puede usarse el mismo dispositivo para programar diferentes plataformas para diversos tipos de robots, agregar o quitar controladores o módulos hardware y que el sistema tenga sólo lo que se vaya a usar. Este tipo de ventajas se abordan en más detalle dentro de la sección 5.4 de discusión.

Hasta el momento se tiene toda la infraestructura y módulos necesarios para implementar la parte hardware del sistema, en la siguiente sección se presenta la arquitectura de la parte software, esta última consta de funciones escritas en lenguaje de alto nivel para que el usuario tenga acceso al hardware de una forma entendible y fácil de usar, dichas funciones se encuentran organizadas en una arquitectura software específica.

4.3. Arquitectura software

La plataforma FPGA debe ofrecer el software necesario para poder acceder y manipular el hardware del sistema. En la figura 4.4 se observa el diagrama funcional propuesto para la arquitectura software. El objetivo es crear una arquitectura software organizada en capas en donde se tengan funciones expresadas en un lenguaje de alto nivel (i.e. lenguaje C) para que el usuario tenga acceso al hardware del sistema, dichas capas se encuentran organizadas en niveles de abstracción (i.e. según la funcionalidad que desempeñen), en donde el nivel de abstracción va creciendo de abajo hacia arriba hasta llegar a las aplicaciones de usuario en donde se tienen funciones totalmente familiares para los programadores de aplicaciones robóticas.

Este trabajo solamente se enfoca a trabajar con las cuatro primeras capas de abstracción: Hardware de Sensores/Actuadores, Controladores/SO, Funciones Básicas y Funciones Intermedias. A partir de esta arquitectura software, se pueden realizar aplicaciones de usuario directamente. Pero también la plataforma queda abierta para realizar una arquitectura software más elaborada y que de soporte a tareas más complejas como la creación de mapas, cinemática de un brazo robot, algoritmos de visión, etc.

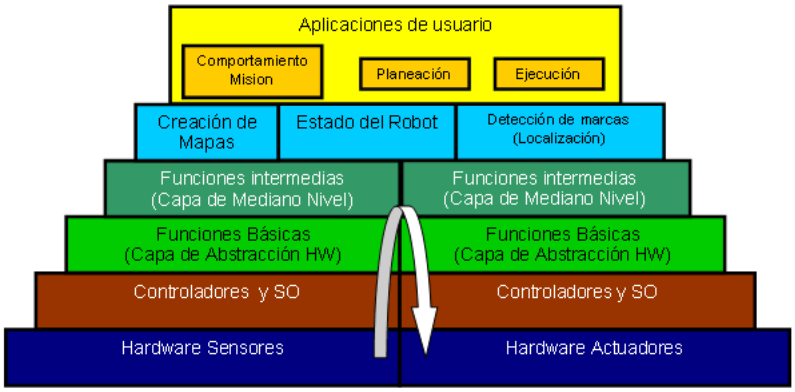


Figura 4.4: Diagrama funcional: arquitectura software. Esta arquitectura esta organizada en capas de abstracción según la funcionalidad que desempeñen las funciones

La idea principal de esta arquitectura software es tener funciones a diferentes niveles de abstracción para que así el usuario tenga disponibles un conjunto de bibliotecas y a partir de ahí se puedan construir directamente aplicaciones de robots o bien construir una arquitectura software más elaborada como ya se había mencionado. Las funciones que son más fáciles de manejar, o por lo menos que son familiares para un programador de robots se encuentran en

las capas Básica e Intermedia. La arquitectura software ofrece funciones en lenguaje C para controlar y manipular los elementos hardware del sistema, los cuales son:

- Robot móvil con una interfaz serial a bajo nivel
- Sensores tipo sonar (que ofrecen anchos de pulso como salida)
- Sensores tipo infrarrojo (seguidor de línea y sensor de proximidad)
- Sensores con interfaz I2C, en este caso un sonar y una brújula
- Brazo robot con interfaz serial
- Co-procesador para procesamiento de imágenes

En el apéndice A.3 se definen y detallan estos elementos extras, también se da una breve explicación de los módulos hardware que se tienen que implementar para poder soportar este tipo de periféricos.

Debido a que la plataforma robótica esta basada en FPGA, es relativamente fácil incorporar nuevos elementos hardware y realizar los controladores tanto a nivel hardware como a nivel software para controlarlos, en este sentido es viable poder incrementar funciones para controlar otro tipo de dispositivos de uso común en robótica.

Cabe señalar que la plataforma software también consta de un SO sobre el cual funcionan todas las bibliotecas, este SO es uno basado en Linux, así que aparte de tener todas las bibliotecas para manipular el hardware se tienen también las funcionalidades que ofrece este tipo de SO (e.g. semáforos, creación de hilos, calendarización, etc.).

A continuación, se detallan cada una de las capas de la plataforma software, también se va ilustrando la funcionalidad de cada capa por medio de un ejemplo, en este caso para el robot Create. Pero antes, en la siguiente sección se da una breve introducción a lo que significa tener un SO Linux en un sistema empotrado.

4.3.1. Linux embebido

Linux embebido (o empotrado) se refiere al uso del sistema operativo Linux en un sistema empotrado, como por ejemplo en PDA's, teléfonos móviles, robots, enrutadores/servidores, dispositivos electrónicos o aplicaciones industriales con microcontroladores y microprocesadores. En el pasado, el desarrollo del software de sistemas empotrados fue llevado a cabo en su mayoría utilizando código propietario escrito en ensamblador. Los desarrolladores debían escribir los controladores para los dispositivos de hardware y las interfaces prácticamente desde cero.

A diferencia de las versiones Linux de las PCs o de los servidores, las versiones empotradas de Linux están diseñadas especialmente para dispositivos con recursos limitados (i.e. sólo los recursos necesarios para la aplicación en específico), como celulares o reproductores mp3. Debido a situaciones como el costo y tamaño, los dispositivos empotrados usualmente tienen mucho menos cantidad de memoria que las computadoras convencionales, y también estos dispositivos generalmente usan memorias FLASH en lugar de discos duros. Dado que los dispositivos empotrados son para funciones específicas, sus desarrolladores optimizan las distribuciones Linux para los sistemas hardware en donde va a correr este SO, esta optimización puede incluir la reducción del número de controladores para dispositivos y algunas aplicaciones software que no se usen.

Tener un SO basado en Linux en el sistema es una buena opción ya que ofrece una capa de abstracción entre el hardware y el software. También, el usuario configura el kernel a la medida de las necesidades del proyecto, teniendo así un kernel tan pequeño como de 1 MB, o bien un kernel que ocupa apenas unos cuantos de cientos de mega *bytes*.

El núcleo de Linux, combinado con un conjunto de algunas otras utilidades de software libre, puede ajustarse dentro del limitado espacio de hardware de los sistemas empotrados. Una

instalación típica de un Linux empotrado ocupa en promedio 2 MB. Existen otros sistemas operativos empotrados como el QNX, LynxOS, Windows CE, Windows NT Embedded, Palm OS.

Linux Empotrado tiene algunas ventajas en relación a otros sistemas operativos empotrados, como pueden ser el código abierto, es pequeño (Windows CE ocupa 21 MB comparado con los 2 MB para Linux Empotrado), puede no tener costos por derechos, es maduro, estable y con un respaldo de millones de usuarios alrededor del mundo.

En el siguiente capítulo se habla específicamente del SO usado para este trabajo, llamado Petalinux (Petalogix, Agosto 2008), que se seleccionó ya que soporta sistemas con procesadores Microblaze y además es una distribución gratuita.

4.3.2. Capas hardware y controladores/SO

Analizando las primeras cuatro capas de la plataforma software propuesta, figura 4.5, se observa que en la capa "Hardware Sensores/Actuadores", se tiene todo el hardware necesario para realizar la comunicación con los diferentes sensores/actuadores de la plataforma (e.g. robot móvil, brazo manipulador, sonar, brújula, etc.). Aquí es donde se implementan los módulos hardware como el módulo UART, módulo I2C, módulo para el brújula, etc. En realidad esta capa pertenece a la arquitectura hardware que se trato en la sección 4.2. Comenzando con el ejemplo, para el caso del robot Create, la capa de "Hardware Sensores/Actuadores" se refiere al módulo UART *full duplex* (que implementa la comunicación serial a una determinada velocidad) para realizar la comunicación hacia el robot. El robot Create posee una interfaz serial por la cual a través de comandos se pueden mandar órdenes al robot, o bien recibir información de sus sensores.

Pasando a la capa de "Controladores y SO", figura 4.5, se tienen las primeras funciones software que tratan con los datos "en crudo" de los periféricos, esto es, en esta capa se transmite y envía información en el lenguaje de los periféricos, es decir, señales eléctricas (generalmente digitales) definidas en una interfaz de comunicación. De esta forma, se tienen funciones software para transmitir o adquirir datos de los diferentes dispositivos hardware. También, en esta capa y las capas superiores se tienen trabajando sobre un SO tipo Linux, el cual ofrece la capa de abstracción entre el hardware y software, además de otras utilidades como creación de hilos, manejo de semáforos, etc.

Figura 4.5: Arquitectura software: capas hardware y controladores/SO. Mediante estas dos capas se obtiene acceso directo al hardware del sistema por medio de funciones en un lenguaje de alto nivel

Siguiendo con el ejemplo, el robot Create tiene un protocolo de comunicación a bajo nivel de tipo serial, por medio del cual se mandan comandos para controlar el movimiento del robot o para recibir información de los sensores, aquí se crean los controladores para manejan a ese módulo hardware, es decir, se crean funciones en C para mandar/recibir *bytes* a través del puerto serial a una determinada velocidad (*baud rate*). En la figura 4.6 se observa una ilustración que

ejemplifica el funcionamiento de estas dos primeras capas. De esta manera, ya se puede realizar una función en C que mande comandos al robot Create para controlar sus movimientos.

En el apéndice C se detallan los pasos para lograr montar un SO basado en Linux dentro del FPGA.

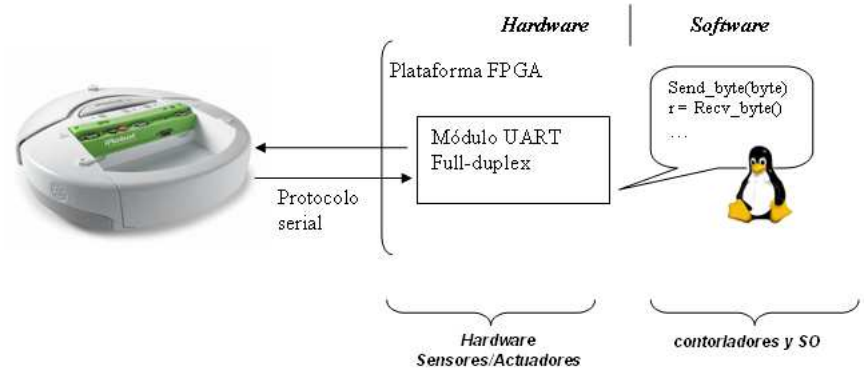


Figura 4.6: Ejemplo de interacción entre las capas Hardware Sensores/Actuadores y Controladores y SO. La capa controladores y SO es la capa de abstracción al hardware del sistema.

En la tabla 4.1 se muestran ejemplos de funciones que se encuentran en la capa de controladores, mientras que en el apéndice B se muestra un listado de todas las funciones para la capa controladores para los diferentes elementos del robot de prueba. En la siguiente sección se define la capa de funciones básicas, la cual tiene un mayor nivel de abstracción en cuanto a que desempeña funciones más complejas.

Función Capa Controladores	**Descripción**
void XUartLite_SendByte (u32 BaseAddress, u8 Data)	Transmite un byte por el módulo UART
u8 XUartLite_RecvByte (u32 BaseAddress)	Recibe un byte por el módulo UART
int XUartLite_Initialize (XUartLite *InstancePtr, u16 DeviceId)	Inicializa el dispositivo UART a sus valores por defecto
void XUartLite_ResetFifos (XUartLite *InstancePtr)	Limpia las FIFOs tanto del transmisor como del receptor
unsigned int XUartLite_Send (XUartLite *InstancePtr, u8 *DataBufferPtr, unsigned int NumBytes)	Envia cierto número de bytes
unsigned int XUartLite_Recv (XUartLite *InstancePtr, u8 *DataBufferPtr, unsigned int NumBytes)	Recibe cierto número de bytes
int XUartLite_IsSending (XUartLite *InstancePtr)	Indica si el buffer del transmisor esta o no vacío
XUartLite_EnableInterrupt (XUartLite *InstancePtr)	Habilita la interrupción del módulo

Tabla 4.1: Funciones de la capa controladores para el robot móvil Create©, básicamente son funciones para el control del módulo serial UART.

4.3.3. Capa funciones básicas

Subiendo a la capa de "Funciones básicas", figura 4.7, en esta capa se tienen funciones software que transmiten/reciben información pero en un lenguaje que el programador de aplicaciones robóticas le sea más entendible, es decir, en esta capa se traduce del lenguaje de los periféricos al lenguaje de los programadores de robots. De esta forma el programador no tiene que preocuparse por el tipo de protocolo que utiliza el periférico para comunicarse, simplemente se fija en el tipo de funcionamiento que quiere desempeñar en el hardware del robot (e.g. avanzar el robot, leer sonares, leer parachoques, etc.).

Siguiendo con el ejemplo del robot Create, en esta capa se tienen funciones que le indican al robot que avance, retroceda, se detenga, gire, pida información del estado de los parachoques, de los botones, del infrarrojo, etc. Las funciones que se encuentren en esta capa hacen uso de las funciones de la capa inferior (Controladores y SO), logrando así aumentar el nivel de abstracción. En la figura 4.8 se observa un ejemplo del funcionamiento de la capa "Funciones básicas" para el robot Create.

Figura 4.7: Arquitectura software: capa Funciones Básicas. En esta capa se tiene acceso al hardware mediante funciones que sean fáciles de usar para la mayoría del los programadores de robots

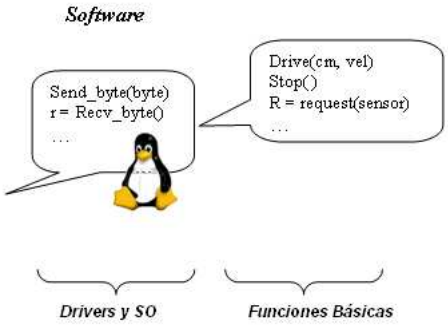


Figura 4.8: Ejemplo de funcionamiento de la capa Funciones Básicas, las funciones en esta capa utilizan las capas inferiores de controladores y el SO para poder comunicarse con el hardware del sistema

Para el robot de prueba Create, se creó una biblioteca de funciones básicas, en la tabla 4.2 se hace una descripción de cada una de estas funciones. Este tipo de bibliotecas puede realizarse para robots móviles compatibles con la plataforma FPGA, ya que una vez que se tengan los controladores para comunicarse con el robot es relativamente sencillo crear funciones con un mayor nivel de abstracción. En la tabla 4.2 se muestra un ejemplo de las funciones que pertenecen a la capa básica para el robot móvil Create. Mientras que en el apéndice B se tiene

un listado de todas las funciones de la capa Funciones Básicas para cada uno de los elementos del robot de prueba, también se da una breve descripción del funcionamiento de cada función.

Función capa Básica	**Descripción**
void send_com(Xuint8 vec[],Xuint8 tam)	Manda comando al Robot Create
void send_com2(Xint16 *vec)	Manda comando al Robot mediante apuntador
void flush_uart()	Limpia el Transmisor del módulo UART
void wait_dist(Xint16 dist)	Espera a que el robot recorra una distancia
void wait_angle(Xint16 ang)	Espera a que el robot gire cierto ángulo
void wait_time(Xuint8 time)	Espera a que pase cierto tiempo
void wait_event(Xuint8 event)	Espera a que pase algún evento
void walk(Xint16 vel)	Aplica la misma velocidad a ambas ruedas
void walk2(Xint16 vel_i, Xint16 vel_d)	Aplica distinta velocidad a cada rueda
void stop_robot(void)	Detiene el robot
void turn(Xint16 vel)	Gira el robot a cierta velocidad sobre su propio eje
void turn_angle(Xint16 angle, Xuint8 vel)	Gira el robot determinado ángulo y a cierta velocidad
void led_manage(Xuint8 op,Xuint8 color,Xuint8 intensity)	Manipula los LEDs de Play y Advanced, así como da intensidad al LED de Power
void play_song(Xuint8 vec[],Xuint8 tam)	Emite serie de sonidos
void send_IR(Xuint8 val)	Recibe información del sensor Infrarrojo
void send_Dout(Xuint8 val)	Envia el valor por el puerto digital del robot
void script(Xuint8 len)	Envia un pequeño script al robot
void play_script()	Ejecuta el script
Xuint8 is_OneByte(Xuint8 pack)	Indica si el paquete a recibir es de uno o dos bytes
void send_sensor(Xuint8 pack)	Manda petición del paquete hacia el robot
Xint16 receive_sensor(Xuint8 num)	Recibe la petición del paquete
Xint16 request_sensor(Xuint8 pack)	Recibe la información del sensor solicitado
Xint16 receive_stream(Xuint8 num)	Recibe información de cierto paquete de forma continua

Tabla 4.2: Funciones de la capa Básica para el robot móvil Create©. Estas funciones ordenan una acción específica al robot, ya sea de movimiento o bien de petición del valor de los sensores

4.3.4. Funciones intermedias

En lo que respecta a la capa de "Funciones Intermedias", figura 4.9, aquí se tienen una serie de comportamientos más elaborados, con cierto nivel de complejidad, aquí se encuentran funciones que sirven como base para las capas superiores. En esta capa se tratan de crear conjuntos de acciones comúnmente utilizadas por el usuario, secuencias de acciones que no se tengan que crear una y otra vez cada que el usuario programe una nueva aplicación, más que bien que se encuentren disponibles y listas para usarse con el simple hecho de agregar la biblioteca correspondiente.

Volviendo al ejemplo del robot Create, en esta capa se tienen funciones que por ejemplo hacen que el robot móvil evite obstáculos, conduzca el robot a cierta distancia y velocidad, que gire cierto número de grados y avance cierta distancia, o bien que espere hasta que algún evento ocurra (e.g. Manden datos por IR, se active algún parachoques, etc). Nuevamente esta capa utiliza funciones de las capas inferiores para lograr un mayor nivel de abstracción.

Estas cuatro capas son la base para que el usuario pueda realizar algoritmos en un lenguaje cómodo y fácil de manejar, que es la finalidad de una plataforma software. Aunque existe la posibilidad de programar a nivel controladores si así lo requiere el programador. Cabe mencionar que esta arquitectura software puede extenderse para que realice tareas más complejas (crear mapas, mover al brazo por medio de cinemática, realizar localización, etc).

Figura 4.9: Arquitectura software: capa Funciones Intermedias. En esta capa se tienen funciones que involucran un conjunto de acciones o comportamientos más elaborados para cierto robot

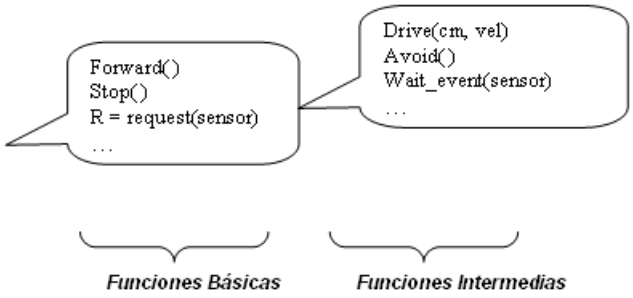


Figura 4.10: Ejemplo de funcionamiento de la capa Funciones Intermedias, esta capa hace uso de las capas inferiores y sirve para realizar tareas o comportamientos que requieren de varias acciones individuales

Nuevamente para el caso de la plataforma de prueba Create, en la tabla 4.3 se muestran las funciones disponibles para la capa intermedia. Mientras que en el apéndice B se encuentra un listado completo de todas las funciones para la capa intermedia que están disponibles para cada elemento del robot.

Función Capa Intermedia	Descripción
void Advance(Xint16 vel)	Avanza el robot a cierta velocidad
void Bumper_stop()	Si se activa algún bumper, detiene el robot
void Bumper_back (Xuint8 dist)	Si se activa algún bumper, retrocede cierta distancia
void Bumper_back_turn (Xuint8 dist, Xint16 angle)	Si se activa algún bumper, retrocede cierta distancia y gira determinados grados
void Bumper_avoid (Xuint8 dist1, Xint16 angle1, Xuint8 dist2, Xint16 angle2)	Si se activa algún bumper, trata de rodear el obstáculo
void Bumper_avoid90 (Xuint8 dist)	Si se activa algún bumper, evade el obstáculo con ángulos de 90 grados
void Bumper_action(Xuint8 side, Xuint8 action)	Dependiendo que lado se active el bumper realiza alguna acción predeterminada
void IR_action(Xuint8 data, Xuint8 action)	Dependiendo el dato que reciba del sensor Infrarrojo, realiza determinada acción
void Detect_OverCurrent()	Detecta si existe un sobreflujo de corriente en las ruedas
void Detect_OverTemp()	Detecta si existe un sobrecalentamiento en la batería del robot Create

Tabla 4.3: Funciones para la capa Intermedia del robot móvil Create. Estas funciones realizan un conjunto de acciones simples

Con esto finaliza la descripción de la plataforma FPGA propuesta en este trabajo, la cual trata de hacer resaltar las ventajas que se obtienen al realizar una plataforma basada en dispositivos FPGA, las ventajas principalmente son a nivel hardware y se ven reflejadas tanto en el desempeño como en la flexibilidad que adquiere el robot. En la siguiente sección se aborda la funcionalidad de la plataforma, es decir, se define el procedimiento que el usuario debe seguir para poder utilizar la plataforma FPGA.

4.4. Funcionalidad de la plataforma FPGA

En esta sección se discute brevemente la funcionalidad de la plataforma FPGA, es decir, la forma en como el usuario tiene que usar la plataforma FPGA y que es lo que el usuario tiene disponible para poder usar la plataforma FPGA.

A nivel hardware, se tiene todo un sistema empotrado totalmente funcional para incorporar un robot móvil con comunicación serial, algunas veces este tipo de robots se usa para propósitos de desplazamiento, es decir, se busca un robot con las características de locomoción deseadas. También se tienen varios recursos hardware para enriquecer la plataforma con otro tipo de elementos, ya sea sensores como sonares, brújulas, IRs, etc. o bien actuadores como el brazo manipulador. Cabe mencionar que todos estos elementos no están sujetos a reglas para poder ser añadidos al sistema, esto es, en cualquier momento el sistema dentro del FPGA puede cambiar sus módulos o añadir nuevos para formar otros tipos de plataforma robótica.

A nivel software, se tiene un soporte como en la mayoría de las plataformas para robots, es decir, un conjunto de bibliotecas con funciones de alto nivel. Estas funciones deben tener la característica de facilitar el desarrollo de aplicaciones al usuario, ofreciendo un lenguaje que él entienda, por ejemplo funciones que directamente ordenen al robot que avance, gire o se detenga. Por otro lado, en cualquier plataforma robótica, tener un SO operando en ella es de gran ayuda. Para el caso de la plataforma FPGA propuesta en este trabajo, se tiene un SO basado en Linux, pero además se pueden montar otro tipo de SO gracias a fabricantes *"third party"*, con esto se tiene un buen abanico de posibilidades a nivel software según las necesidades del proyecto.

Cabe mencionar que la arquitectura software queda abierta para realizar una arquitectura más elaborada y que de soporte a tareas más complejas como creación de mapas, cinemática del brazo, algoritmos de visión, etc.

En general, los pasos que debe seguir el usuario para usar la plataforma FPGA en una aplicación robótica son los siguientes:

1. Definir los elementos de la parte hardware utilizando las herramientas software del fabricante (opcional).
2. Programar el FPGA con el archivo bitstream
3. Configurar el kernel del SO Linux que correrá en el FPGA (opcional)
4. Programar las aplicaciones de robots en lenguaje C utilizando las bibliotecas del apéndice B y utilidades del SO
5. Bajar la imagen del SO Linux a la memoria DDR SDRAM de la tarjeta
6. Generar un programa *bootloader*[2] y guardar el archivo bitstream y la imagen del SO Linux para guardarlo de forma permanente en la memoria FLASH de la tarjeta y de esta forma sólo se tiene que prender la tarjeta para que todo arranque de forma automática (opcional)

El principal inconveniente es que se debe contar con una persona que esté familiarizada en manejar FPGAs así como sus herramientas de desarrollo, ya que la manipulación de la arquitectura hardware y la compilación de aplicaciones usando Linux empotrado no es trivial.

Una solución idónea y viable es ofrecer un conjunto de herramientas y utilidades que permitan automatizar el proceso, es decir, ofrecer al usuario un conjunto de *bistreams* los cuales contengan diferentes arquitecturas hardware para soportar varios tipos de robots, sensores o actuadores. Mientras que por el lado software, ofrecer también un conjunto de configuraciones de *kernels* que se adecuen a las diferentes arquitecturas hardware, además de tener utilidades para que el usuario programe y compile fácilmente y pueda bajar la parte software al FPGA.

En resumen, se tiene una plataforma FPGA flexible para implementar diferentes tipos de robots "híbrido", es decir, se puede controlar un robot móvil comercial que tenga una interfaz a bajo nivel, luego a este robot se pueden añadir una buena variedad de elementos comúnmente utilizados en robótica móvil y que sean compatibles con la plataforma FPGA. Además se proporciona una arquitectura software que tiene funciones base para el manejo de todos los elementos hardware y quede abierta para poder extender su funcionalidad ya sea para añadir nuevos elementos a la plataforma o bien para crear funciones más elaboradas que realicen tareas complejas.

En el siguiente capítulo se describen los detalles de implementación así como las pruebas realizadas y los resultados obtenidos tanto individualmente como en conjunto para validar la plataforma robótica FPGA.

[2] *Bootloader* es un programa sencillo diseñado exclusivamente para preparar todo lo que necesita el sistema operativo para funcionar. En este caso se encargaría de cargar el SO desde la FLASH a la memoria DDR SDRAM.

Capítulo 5

Experimentos y resultados

En esta sección se presentan los experimentos realizados para comprobar el correcto funcionamiento de la Plataforma FPGA así como resaltar los resultados y ventajas que se obtienen al usar este tipo de dispositivos. También se presentan brevemente los pasos que se siguieron para la implementación de la plataforma FPGA.

5.1. Implementación

En este apartado se presentan a grandes rasgos los pasos de implementación, en el apéndice A se habla más en detalle del proceso de implementación utilizando las herramientas de desarrollo.

5.1.1. Material y equipo de implementación

En esta sección se presentan los detalles de la tarjeta FPGA usada para este trabajo. Cabe señalar que los conceptos y aportaciones dadas en este trabajo pueden ser usadas para controlar a robots móviles que tengan una interfaz serial o bien algún tipo de protocolo de comunicación similar a bajo nivel, además se pueden incorporar nuevos sensores que tengan interfaces del tipo SPI, I2C, serial *full-duplex* y serial *half-duplex*.

En las figuras 5.1,5.2 y 5.3 se observa el robot móvil de prueba y se van localizando cada uno de sus componentes. El robot móvil de prueba fue construido con diferentes elementos y que incluye la plataforma FPGA. A continuación se enlistan los elementos del robot de prueba:

- Tarjeta FPGA Spartan 3E 1600, SP3E1600E MicroBlaze Edition (Xilinx, Diciembre 2008)
- Robot móvil Create (iRobot, Septiembre 2008)
- Brazo Manipulador: Smart Arm AX-12 (CrustCrawler, Agosto 2008)
- 5 sonares: 3 PING Ultrasonic (Parallax, Septiembre 2008), 1 SRF02 (Acroname, Diciembre 2008) y 1 SRF05 (Acroname, Diciembre 2008)
- 1 brújula: CMPS03 (Acroname, Diciembre 2008)
- 1 sensor IR de proximidad: GP2D15 (Lynxmotion, Diciembre 2008)
- 1 sensor IR seguidor de línea (Lynxmotion, Diciembre 2008)
- 1 batería de 9.6 v con 2000 mA/h (para brazo AX-12)
- 1 batería de 5 v con 2500 mA/h (para tarjeta FPGA y sensores)

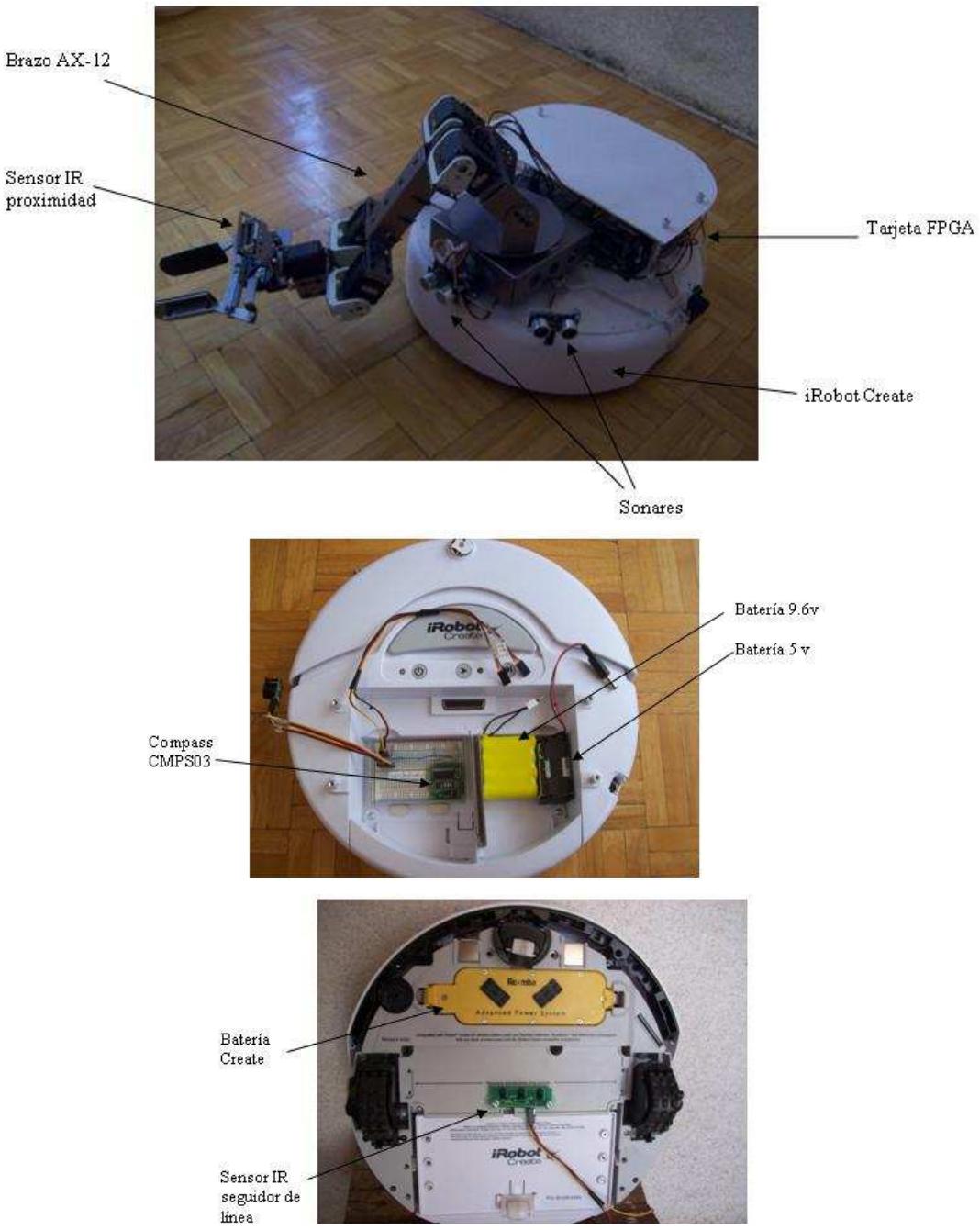


Figura 5.1: Robot móvil de prueba y localización de sus componentes (i.e. tarjeta FPGA, sensores infrarojos, brujula, brazo manipulador y baterías

Figura 5.2: Vista general del robot móvil de prueba

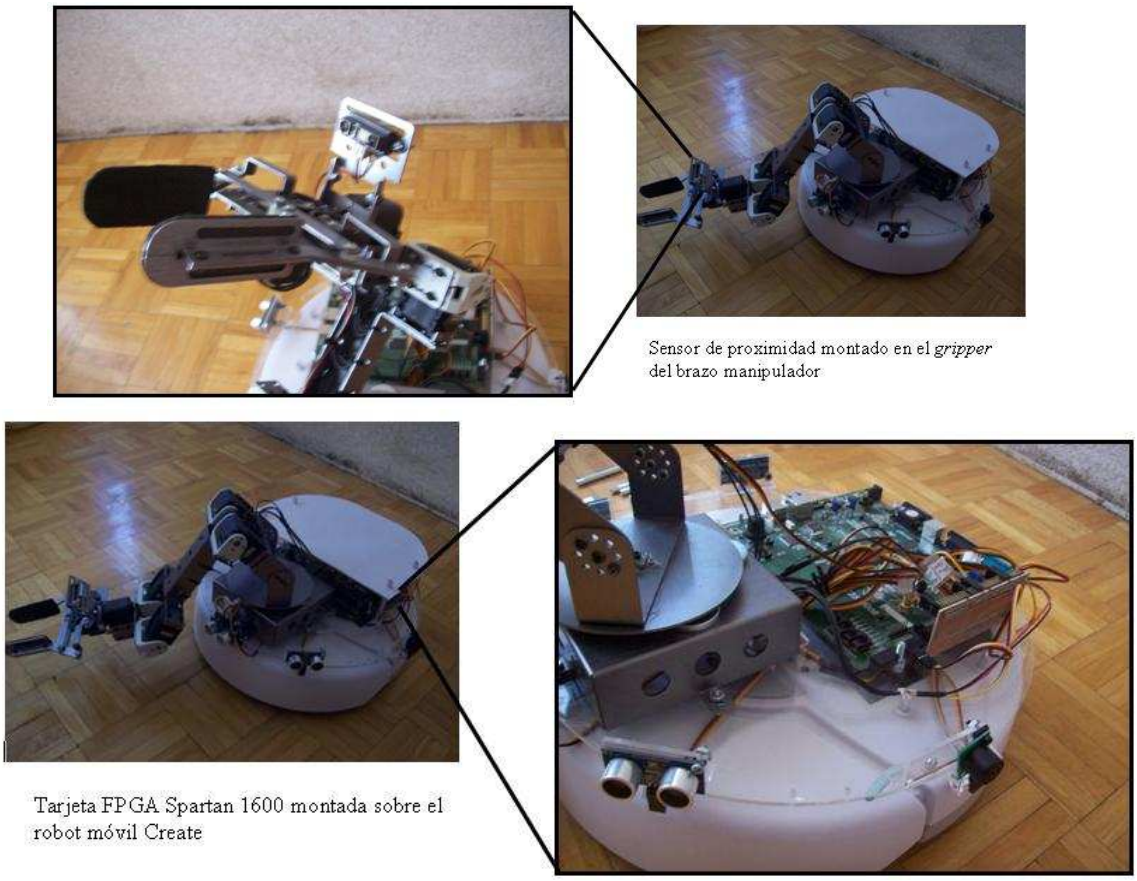


Figura 5.3: Robot móvil de prueba, acercamientos a diferentes partes del robot

Por lo que respecta al dispositivo FPGA usado, se trata de una tarjeta FPGA Spartan 3E 1600, figura 5.4, la cual tiene como características principales las siguientes:

- FPGA Spartan 3E con una equivalencia en compuertas de 1.6 millones
- Implementación del procesador Microblaze y del micro-controlador PicoBlaze

Figura 5.4: Tarjeta FPGA SP3E1600E MicroBlaze Edition, de la empresa Xilinx. Figura tomada de Xilinx (Diciembre 2008)

- DDR-SDRAM de 64 MB, FLASH de 16 MB, FLASH SPI 16 Mb
- LCD display, Puerto VGA, puerto PS/2, 10/100 Ethernet
- Dos puertos RS232, DTE y DCE
- Codificador rotatorio, LEDs, interruptores, botones y puertos de expansión para propósito general.

Cabe mencionar que para la plataforma FPGA de este trabajo se uso una tarjeta comercial (Spartan 3E 1600) de la compañía Xilinx, pero se puede construir una tarjeta específica (como en la figura 4.1) utilizando un subconjunto de las capacidades y periféricos de las tarjetas de desarrollo comerciales. Construir este tipo de tarjetas está fuera del alcance de esta tesis, aunque es bastante viable realizar una.

Finalmente, la computadora usada para desarrollar la plataforma FPGA y los programas instalados en ella son:

- Computadora con procesador Intel Centrino Duo, 2.0 GHz, Windows XP-32 bits.
- 2 GB de memoria RAM
- 120 GB disco duro
- Herramientas de desarrollo EDK 9.2, ISE 9.2, para el diseño de la plataforma FPGA
- MATLAB©ver 7.3 y Xilinx System Generator©9.2 para el desarrollo del co-procesador de imágenes
- Herramienta VMware para crear una máquina virtual con CentOS Linux para realizar la compilación-cruzada

5.1.2. Herramientas de implementación

Los sistemas empotrados son complejos de diseñar. Hacer el diseño de las partes tanto hardware como software son proyectos completos por si mismos, y fusionar estas dos partes de tal manera que puedan formar un sistema empotrado completo requiere un desafío mayor. Ahora

bien, agregar un FPGA a todo esto y hacer que el sistema funcione como se debe se vuelve aún más confuso y una tarea difícil.

Para simplificar el proceso de diseño de sistemas empotrados usando FPGAs, Xilinx ofrece las siguientes soluciones:

- ***ISE (Integrated Software Enviroment©).*** Es la herramienta fundamental para el diseño de sistemas lógicos en FPGAs. El diseño de proyectos con FPGAs puede ser una tarea muy envolvente, ISE tiene herramientas software que permiten ayudar al diseñador con este proceso. Varias utilidades como el Editor de restricciones para asignación de pines y de área, herramientas automáticas para el enrutado y la localización de componentes, además ISE contiene una de las utilidades más importantes que es el sintetizador XST, el cual permite crear un netlist desde un lenguaje HDL que contenga toda la información del diseño en términos de los recursos del FPGA.

- ***EDK (Embedded Development Kit©).*** EDK es un conjunto de herramientas y *cores* IP que permiten el diseño de sistemas empotrados completos basados en procesadores para ser implementados en dispositivos FPGA. Dentro de las herramientas que posee está XPS, el cual es un entorno de desarrollo o una GUI usada para diseñar la parte hardware principalmente del sistema empotrado. Mientras que la herramienta SDK es un entorno de desarrollo, complementario a XPS, que es usado para la creación y verificación del software escrito en C/C++ del sistema empotrado. SDK fue construido bajo el marco de código abierto de Eclipse. Otros componentes de EDK son:

  - Hardware IP para los procesadores empotrados de Xilinx (Microblaze y PowerPC)
  - Controladores y bibliotecas para el desarrollo del software
  - Compilador GNU y depurador C/C++ para el software desarrollado

El proceso de implementación se divide en tres principales pasos: 1) Desarrollo de la arquitectura hardware, 2) Desarrollo de las bibliotecas software y 3) Montar Linux en el FPGA, los cuales se describen en las secciones 5.1.3, 5.1.4 y 5.1.5 respectivamente.

5.1.3. Desarrollo de la arquitectura hardware

Como ya se había mencionado, la plataforma FPGA es un sistema empotrado basado en el procesador Microblaze, el cual es altamente configurable y permite obtener un balance entre el desempeño y área, lo cual hace que se tenga un sistema flexible. Los pasos que se tienen que realizar en la herramienta EDK para desarrollar la arquitectura hardware son los siguientes:

- Colocar todos los módulos hardware (*cores*) que tendrá la arquitectura hardware, módulos como: procesador, buses, memorias, controladores, co-procesadores, módulos de usuario, etc.

- Conectar a nivel general todos los componentes por medio de los *buses*.

- Realizar las conexiones de las señales de E/S de cada componente y decidir que señales serán externas al FPGA.

- Asignar el espacio de memoria para cada periférico, este espacio de memoria es necesario para que el procesador pueda comunicarse con sus periféricos.

Una descripción en detalle de la implementación de la arquitectura hardware en EDK puede observarse en el apéndice A.
Muchos de los módulos hardware (*cores*) de la plataforma FPGA propuesta vienen incluidos por

defecto con la herramienta EDK, algunos otros se adquieren comprando una licencia para poder utilizarlos y también existe la posibilidad de que el usuario realice sus propios módulos hardware e incluirlos en el sistema Microblaze como un *core* más. Así que otro aspecto importante de la arquitectura hardware es el desarrollo de módulos hardware personalizados así como el desarrollo de co-procesadores. El hecho de implementar módulos propios hace que la arquitectura sea hecha a la medida para las necesidades del proyecto además de mejorar el rendimiento del sistema por medio de co-procesadores. A continuación se muestran los módulos hardware implementados para este trabajo.

Sonares

Los sonares son un tipo se sensor bastante utiizados para los robots móviles en sus tareas de navegación, existe una buena variedad de este tipo de sensores, aunque básicamente la mayoría entrega dos tipos de señal: algunos entregan un ancho de pulso proporcional a la distancia del objeto y otros entregan ya la distancia al objeto en una medida estándar (e.g. cm) pero por medio de un protocolo serial. La plataforma propuesta puede soportar ambos tipos de sensor, ya que se puede implementar el hardware que hace la conversión de un ancho de pulso a una medida en centímetros, o bien se puede implementar el protocolo de comunicación para realizar la interfaz con el sensor (e.g. I2C, SPI, etc).

Para este trabajo se emplearon tres distintos tipos de sonares: SRF05, PING y SRF02. La descripción para cada sonar se puede encontrar en el apéndice A.

En la figura 5.5 se observa el diagrama a bloques del periférico completo para trabajar con el sonar SRF05. La funcionalidad este módulo es mandar un pulso a la entrada de *reset*, para que automáticamente se mande un pulso de salida para activar la señal de *burst* del sonar, una vez que la señal emitida por el sonar rebota y el eco de entrada (ancho de pulso) llega, ahora comienza el proceso de conversión, dando finalmente como resultado la salida 'salcm' (resultado de la distancia en cm). También se observa la interconexión del módulo para poder ser conectado al bus PLB del procesador y de esta forma pueda ser controlado por software.

De forma análoga, en la figura 5.6 se observa el diagrama a bloques del periférico completo para trabajar con el sonar PING ultrasonic. La principal diferencia es que se implementa una lógica de salida mediante un *buffer* para que por el mismo pin se implementen las señales de entrada y salida de eco.

Por lo que respecta al sonar SRF02, ya que trabaja con el bus I2C, este tipo de sonar se presenta en la siguiente sección.

Como puede observarse en los diagramas hardware para los sonares, existen unos nombres llamados 'slv_reg', los cuales son registros que pueden accederse por medio de funciones software, estos registros son los que van ayudar a activar y leer el resultado de los sonares a través de aplicaciones de usuario. Las funciones disponibles para este tipo de periféricos se encuentran en la tabla B.4.

Bus I2C

Muchos de los dispositivos en Robótica móvil funcionan bajo el protocolo I2C. En un bus I2C puede haber dispositivos maestros y esclavos, solamente los maestros pueden iniciar una comunicación. A continuación se describe brevemente las reglas de comunicación para este bus, éstas reglas son las que se tienen que implementar en un *core* para poder comunicar a los dispositivos:

1. La condición inicial de bus libre, es cuando ambas señales SDA y SCL están en estado alto. En este momento cualquier dispositivo maestro puede ocupar el bus, estableciendo la condición de inicio. Esta condición de inicio se da cuando la línea SDA se pone en estado bajo y la señal SCL permanece en alto.

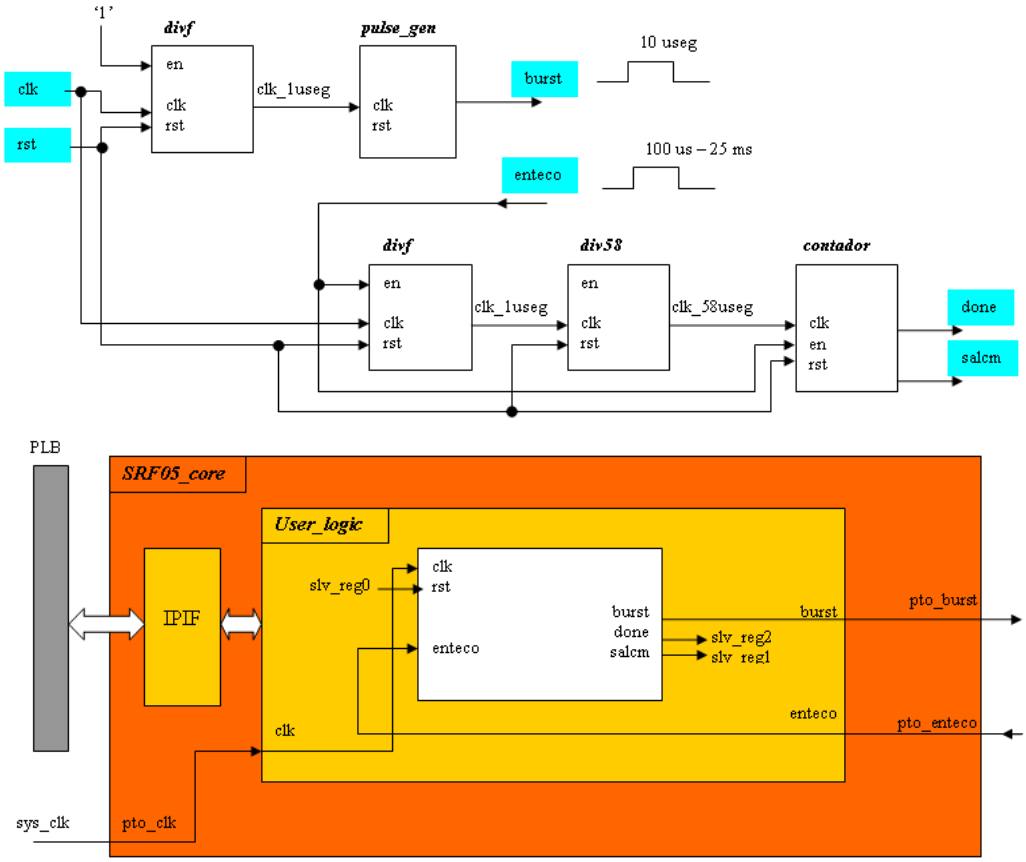


Figura 5.5: Diagrama para el módulo hardware del sonar SRF05. Módulo de conversión que obtiene la distancia en centímetros apartir del ancho de pulso (arriba) y módulo de interconexión con el bus PLB (abajo) para integrar el módulo de conversión al sistema Microblaze

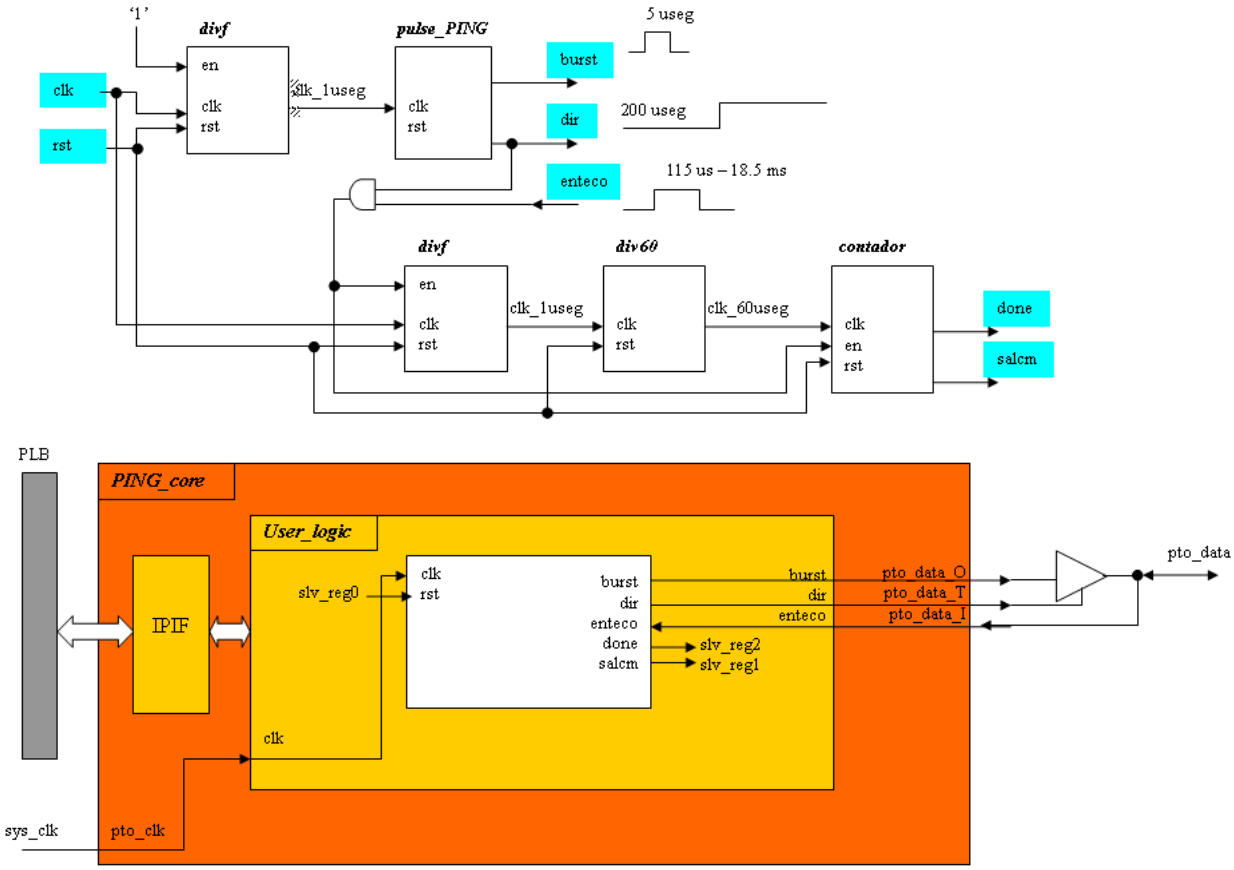


Figura 5.6: Diagrama del módulo hardware para el sonar PING Ultrasonic. Módulo de conversión que entrega la distancia en centímetros apartir de un ancho de pulso(arriba) y módulo de interconexión con el bus PLB (abajo) para poder integrar el módulo de conversión en el sistema Microblaze

2. El maestro se dirige al dispositivo, enviando un byte que contiene 7 bits (A7-A1) de dirección del dispositivo y un octavo bit (A0) que indica si la operación al dispositivo es de lectura o escritura.

3. La petición de dirección es comparada por cada dispositivo esclavo con su propia dirección, si ambas coinciden, el esclavo responde enviando un bit de ACK indicando al dispositivo maestro que esta en condiciones de comunicarse.

4. Después comienza el intercambio de información entre los dispositivos. El maestro manda la dirección del registro que desea leer/escribir. El esclavo responde con otro ACK.

5. Ahora el maestro comienza a leer o escribir bytes de datos desde/hacia el dispositivo esclavo. Cada byte leído/escrito va acompañado de su correspondiente bit ACK.

6. Cuando se han transmitido todos los datos, la comunicación debe finalizar con una condición de paro dejando así el bus libre nuevamente, la condición de paro se da cuando la señal SDA se pone a estado alto, mientras que la señal SCL se encuentra en estado alto también.

De esta manera la trama de datos que usa el bus I2C sería la siguiente:

| start | A7 A6 A5 A4 A3 A2 A1 | R/W | ACK | ... DATA ... | stop | idle |

Para este trabajo se emplearon dos dispositivos que trabajan mediante bus I2C: la brújula CMPS03 y el sonar SRF02. La descripción para estos dos dispositivos se da en la sección **??**

La secuencia de funcionamiento que se debe implementar a nivel software el sensor SRF02 es la siguiente:

- Escribir la dirección del dispositivo 0x0E para establecer comunicación.
- Escribir al registro de comandos (0x00) el modo de medición que se desea: uS (0x50), cm (0x51), in (0x52)
- Esperar un retardo de 70 useg aproximadamente para esperar a que el eco regrese al sensor.
- Leer los registros 0x02 y 0x03 para el resultado de la medición, un valor de 0 devuelto indica que ningún objeto fue detectado.

La secuencia de funcionamiento que se debe implementar a nivel software para la lectura del sensor CMPS03 es la siguiente:

- Escribir la dirección del dispositivo 0xC0 para establecer comunicación
- Leer los registros 0x01 o bien 0x02 y 0x03 para obtener el resultado

Para validar estos dos dispositivos, se realizó una aplicación software en donde mediante un menú el usuario puede seleccionar la lectura del SRF02, la lectura de la brújula o bien calibrar la brújula. Las funciones usadas para controlar estos dispositivos aparecen en las tablas B.4 y B.5.

Controlador UART *half-duplex*

Para controlar el brazo manipulador AX-12, es necesario implementar un módulo serial UART pero de tipo *half-duplex*[1], ya que es el protocolo de comunicación que utiliza este tipo de dispositivo.

El módulo UART16550, es un *core* que implementa la comunicación serial tipo *full-duplex* con la característica principal de que la velocidad de transmisión es seleccionada por software, escribiendo a un registro, esto permite generar velocidades de transmisión que pueden no ser estándares, esto ayuda ya que el brazo funciona a una velocidad de 1 Mbps.

[1] Una comunicación *half duplex* significa que el método o protocolo de envío de información es bidireccional pero no simultáneo. Tanto el transmisor como el receptor se encuentren por el mismo hilo de comunicación.

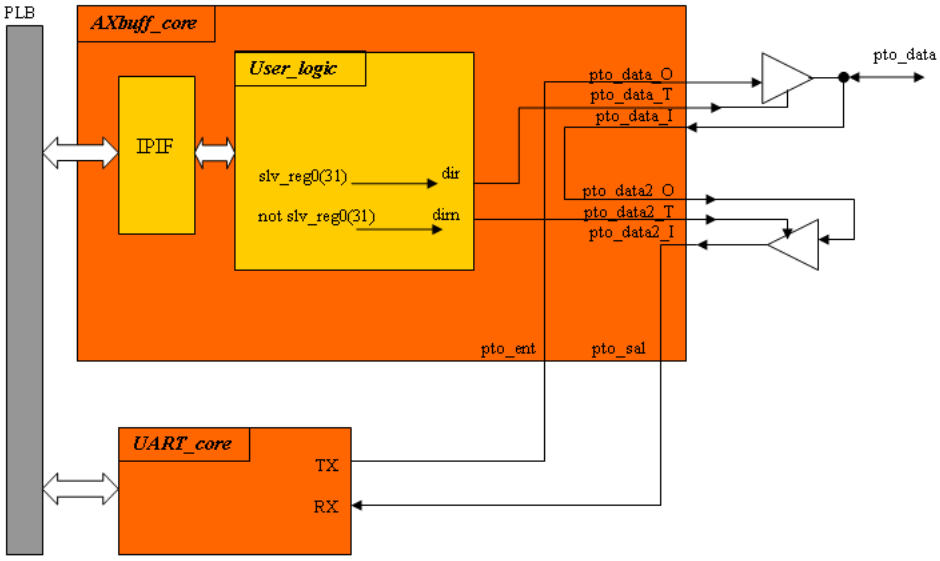


Figura 5.7: Diagrama de conexión para el módulo hardware que controla el Brazo AX-12, se utiliza un módulo UART *full-duplex* que mediante lógica de buffers tri-estado se convierte en un módulo UART *half-duplex* necesario para realizar la comunicación con el brazo AX-12

Ya que el controlador usado es de tipo *full-duplex* es necesario implementar una lógica de *buffers* tri-estado para convertir la comunicación a una de tipo *half-duplex*. En la figura 5.7 se observa el diagrama para la lógica de *buffers* y la manera en como se interconecta con el core UART16550. Las funciones para controlar este dispositivo son las que aparecen en la tabla B.7.

Co-procesador para imágenes

La tarea del co-procesador para este trabajo es implementar un módulo hardware que tenga múltiples procesadores de convolución trabajando en paralelo, con la finalidad de realizar el proceso de convolución más rápido, aprovechando el paralelismo del FPGA. La convolución en 2-D es una operación básica en diversos algoritmos de bajo y mediano nivel de procesamiento de imágenes. En particular sirve para el filtrado y extracción de contornos u otras características.

La herramienta utilizada para implementar este *core* se llama *Xilinx System Generator*©(XSG)[2], la cual es otra de las herramientas de Xilinx, especialmente diseñada para trabajar con aplicaciones de procesamiento digital de señales (DSP). La metodología es implementar el coprocesador en XSG y después importarlo a EDK como una caja negra para poder utilizarlo como periférico. El tipo de *bus* utilizado es el FSL diseñado especialmente para la comunicación de co-procesadores en un sistema Microblaze.

Este co-procesador tiene la característica de ser parametrizable por el usuario, dentro de estos parámetros se tiene principalmente el tamaño de la imagen, el tamaño de la máscara, la representación de los coeficientes de la máscara, entre otros. Esto es de bastante utilidad ya que el usuario puede aplicar cualquier tipo de máscara de convolución a la imagen y decidir el formato con el que quiere representar a los coeficientes de la máscara.

La idea central del core es tener múltiples "módulos de convolución" funcionando en paralelo, con el objetivo de disminuir el número de accesos a memoria. Procesando el mismo pixel para diferentes posiciones de la ventana de convolución, esto acelera el proceso global de con-

[2]XSG es un ambiente de diseño integrado (IDE) a nivel sistema para FPGAs, que utiliza a Simulink de MATLAB©como entorno de desarrollo y se hace presente en forma de *blockset*.

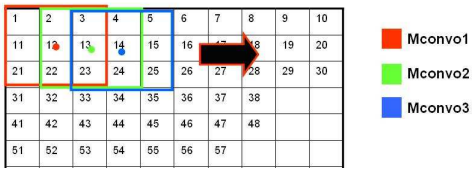


Figura 5.8: Esquema general del proceso de convolución con múltiples módulos trabajando en paralelo con la finalidad de acelerar el proceso de convolución en una imagen

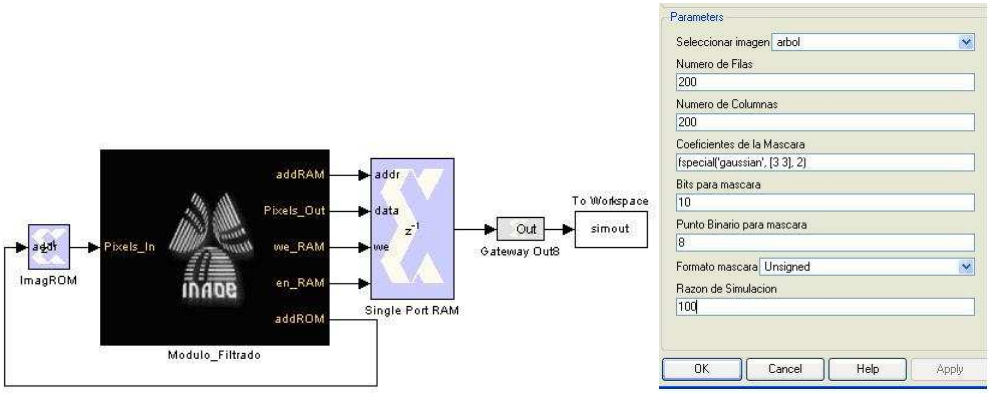


Figura 5.9: Subsistema del co-procesador para imágenes y su cuadro de diálogo, el cual se utiliza para configurar de forma automática la arquitectura hardware dentro del subsistema. Tanto la interfaz como la arquitectura fueron realizadas con la herramienta Xilinx System Generator©y MATLAB©

volución obteniendo así una imagen de salida más rápido. En la figura 5.8 se ilustra la idea de funcionamiento.

En la arquitectura propuesta para este *core*, el número de módulos de convolución es igual al número de columnas de la máscara. Este esquema permite mejorar el rendimiento de la arquitectura en un factor de 'n', donde n es el número de columnas de la máscara. Los módulos de convolución están compuestos únicamente de un multiplicador y un acumulador, que son los elementos escenciales en el proceso de convolución.

Para que la arquitectura sea parametrizable se utilizaron dos conceptos importantes de MATLAB©: generación dinámica de bloques y enmascaramiento de subsistemas, ambos conceptos se discuten en el apéndice A. La figura A.13 muestra el co-procesador como una caja negra así como el cuadro de diálogo que lo configura.

En la figura A.14 se observa la arquitectura dentro del subsistema y de como ésta va cambiando conforme el usuario selecciona parámetros distintos. En este ejemplo se observa la arquitectura generada automáticamente para a) una máscara de convolución de 3x3 y b) una máscara de 5x5.

Una vez que se tiene el core configurado a las necesidades del usuario, el siguiente paso es importarlo como periférico al sistema Microblaze. Esto se realiza por medio de una utilidad de la herramienta EDK y que funciona por medio de un *wizard* que va llevando paso a paso la configuración del co-procesador. De esta manera una vez hechas todas las conexiones necesarias y tener el co-procesador listo en el sistema Microblaze, sólo resta colocar los datos de entrada y leer los resultados mediante dos instrucciones simples que se usan desde la aplicación software, estas instrucciones junto con otras funciones con un mayor nivel de abstracción son las que se muestran en la tabla B.8.

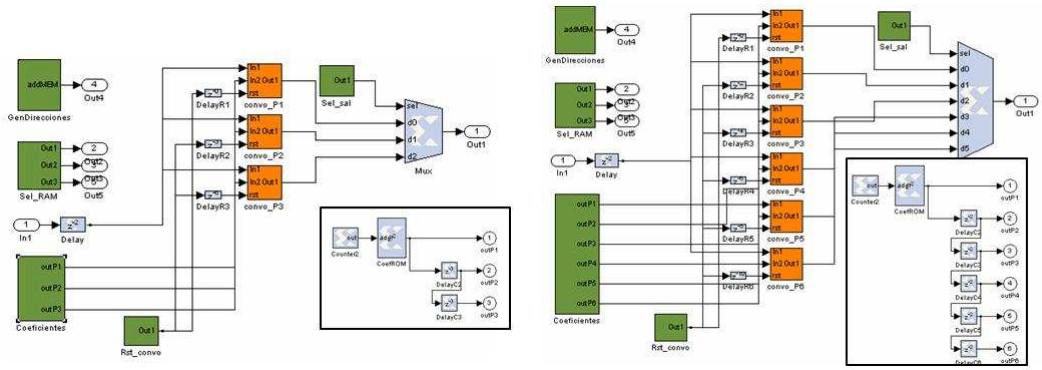


Figura 5.10: Efecto de la programación dinámica de bloques en el core para imágenes. Arquitectura para una máscara de 3×3 (izquierda) y arquitectura para una máscara de 5×5(derecha)

5.1.4. Desarrollo de bibliotecas software

Una vez que se ha concluido la realización de la parte hardware del sistema, ahora se tiene que realizar la parte software del mismo, es decir, los programas de usuario que estará corriendo el procesador. Antes de crear cualquier aplicación de usuario es necesario generar toda una infraestructura software que se comunique a bajo nivel con la plataforma hardware, este puente entre en hardware y software se llama *Board Support Package* (BSP).

El BSP es una colección de archivos que definen, para cada procesador, los elementos hardware del sistema. El BSP también contiene información de los elementos software, tal como los controladores para cada periférico, bibliotecas, rutinas de interrupción, dispositivos estándar de E/S, y otras características relacionadas. El BSP también permite cargar un SO sobre la plataforma hardware.

Una vez que el BSP es generado por la herramienta EDK el siguiente paso es crear las aplicaciones de usuario. Las aplicaciones de usuario se escriben en lenguaje C y utilizando las bibliotecas del apéndice B, después se pueden ir construyendo funciones y bibliotecas con un mayor nivel de abstracción. A continuación se muestra un ejemplo de aplicación de usuario, figura 5.11 el cual hace que el robot se mueva hacia adelante o se detenga según lo que se seleccione en el menú.

5.1.5. Desarrollo para montar Linux en el FPGA

La distribución que se usa para este trabajo es 'Petalinux' de la empresa Petalogix (Petalogix, Agosto 2008). Petalinux es un puerto del SO uClinux especialmente diseñado para trabajar con el procesador Microblaze de Xilinx, aunque también soporta otras arquitecturas como Coldfire, Blackfin y ARM7; además es una distribución gratuita. uClinux es un proyecto destinado a portar Linux a dispositivos sin unidades de manejo de memoria (MMU), esto implica que el espacio de memoria es continuo y fijo, no existe protección de memoria, así que cada proceso del kernel o de usuario puede acceder cualquier espacio de memoria. Aunque las nuevas versiones del procesador Microblaze ya soportan la unidad MMU.

En general, para montar un sistema operativo Linux en un FPGA se necesitan cuatro cosas:

1. La plataforma hardware que funciona sobre una tarjeta FPGA.

2. El BSP necesario para la plataforma hardware.

3. La distribución de Linux, en este caso Petalinux.

```
// ---- CODIGO EN LIBRERIAS (Capa Basica) ----

Xuint8 full_mode[]={128,132};

//Mandar un vector de comandos
void send_com(Xuint8 vec[],Xuint8 tam)
{
int i;
for(i=0; i<tam; i++)
 XUartLite_SendByte(XPAR_RS232_DTE_BASEADDR, vec[i]);
}

//Camina hacia adelante o atras
//vel esta dada en mm/s, rango [-500  500]
void walk(Xint16 vel)
{
Xuint8 lowb,highb;

lowb = vel;
highb = vel >> 8;

Xuint8 vec[]={145,highb,lowb,highb,lowb};
send_com(vec,sizeof(vec));
return ;
}

//--------------
//Detiene el robot
void stop_robot(void)
{
Xuint8 vec[]={145,0,0,0,0};
send_com(vec,sizeof(vec));
return;
}
//---- PROGRAMA PRINCIPAL ----
int main (void) {

print("\r\nPrograma Prueba \n");
print("\r\n1.Avanza \r\n2.Detener \r\nX.Salir \n");

send_com(full_mode,sizeof(full_mode)); //full mode

while(flag==0){
        print("\r\nOpcion: ");
        op=XUartLite_RecvByte((void *)XPAR_RS232_DCE_BASEADDR);
        switch(op)
                {
                case '1':
                        walk(150);
                        break;
                case '2':
                        stop_robot();
                        break;
                default:
                        flag=1;
                        break;
                }
        }//fin del while

stop_robot();
print("\r\nFin del programa");
```

Figura 5.11: Ejemplo de aplicaciones de usuario. Uso de las funciones de biblioteca para controlar el robot. En este caso se controla la dirección hacia donde el robot avanza

4. El MicroBlaze GCC *tool-chain*, el cual incluye todo lo necesario para construir el kernel de uClinux en un ambiente Linux.

El apéndice C tiene una descripción más detallada acerca de como montar un SO basado en Linux en un FPGA.
En esta sección se describió el proceso de implementación, a continuación se presentan las pruebas y los resultados obtenidos, los cuales muestran algunas de las ventajas que se pueden lograr utilizando los dispositivos FPGA para el desarrollo de plataformas robóticas. A continuación se describen los criterios de evaluación seguidos así como el ambiente para las pruebas del robot.

5.2. Criterios de evaluación y ambiente de prueba

Los experimentos realizados están basados en algunas pruebas con elementos individuales de la plataforma FPGA (e.g. sonares, brújula, etc.) con el objetivo de resaltar las ventajas obtenidas al implementar tareas en hardware.

Los criterios de evaluación para este trabajo no están enfocados del todo sobre los experimentos, más bien se enfocan en realizar comparaciones cuantitativas y cualitativas entre la plataforma FPGA propuesta y tarjetas robóticas controladoras basadas en microcontroladores, estos últimos comúnmente utilizados en los robots móviles, aunque también existen tarjetas basadas en procesadores. Estas comparaciones destacan las ventajas y desventajas que ofrecen los FPGAs contra los microcontroladores. Las comparaciones están basadas en ciertas características, las cuales fueron seleccionadas de acuerdo a los factores que hacen que una plataforma robótica sea atractiva para algún proyecto y además de lo que un usuario buscaría cuando va a utilizar o adquirir una tarjeta de este tipo. Estas características fueron el resultado de una encuesta realizada a veinte personas relacionadas con robots móviles, entre estas personas se tienen: académicos, alumnos y algunos doctores. Estas características son:

- Desempeño: estimación de MIPS (millones de instrucciones por segundo[3])
- Software de programación
- Soporte de sensores y actuadores
- Protocolos de comunicación
- Opciones de expansión hardware
- Precio, tamaño y peso
- Versatilidad para agregar nuevos periféricos (aceleradores hardware, reconfiguración, etc.)
- Tipo de alimentación

En cuanto al ambiente de prueba es de tipo interior y consta de un área de $2{,}76 \times 2{,}76$ metros, rodeada por tres paredes. Dentro del área de prueba se tienen distintos aditamentos dependiendo de la prueba a realizarse. Por ejemplo, para la prueba del laberinto se tiene una pista dibujada con cinta y unas marcas que indican en donde hay un punto en donde el robot debe regresarse, o debe parar, o debe girar. Para la prueba del seguidor de pared se tienen plataformas hechas con cajas de cartón para sostener los objetos que transportará el robot mediante el brazo manipulador. Los objetos son pequeñas esferas (de 4 y 5 cm de diámetro). Un diagrama del ambiente de prueba se muestra en la figura 5.12.

[3]MIPS (Millones de Instrucciones por segundo). Es una forma de medir la potencia de los procesadores. Sin embargo, esta medida sólo es útil para comparar procesadores con el mismo juego de instrucciones y usando *benchmarks* que fueron compilados por el mismo compilador y con el mismo nivel de optimización.)

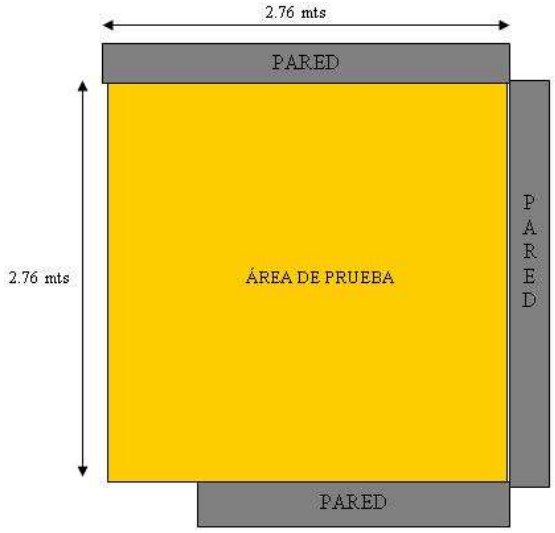


Figura 5.12: Ambiente de prueba para la plataforma robótica basada en FPGA. Consiste en un área cuadrada rodeada por tres paredes

5.3. Experimentos

A continuación se describen los experimentos realizados así como los resultados obtenidos de cada uno de ellos.

5.3.1. Incorporar protocolos de comunicación

En este experimento se hace uso de la brújula CMPS03, la cual utiliza el protocolo I2C, cabe señalar que este sensor también proporciona una salida PWM que puede ser utilizada en lugar del protocolo I2C. La brújula proporcionar datos de orientación del robot.

Con este experimento se muestra la flexibilidad del FPGA para poder incorporar diversos tipos de protocolos de comunicación usualmente utilizados en los dispositivos para robótica (e.g. I2C, SPI). Existe una gran variedad de códigos fuente en la red descritos en algún lenguaje HDL (e.g. Verilog y VHDL) para protocolos de comunicación y que se encuentran disponibles de manera gratuita, o bien, se puede optar por realizar códigos HDL de forma personalizada. Por otro lado, EDK tiene las herramientas necesarias que permiten incorporar estos protocolos como perféricos a un sistema empotrado Microblaze. De esta manera se tiene un buen abanico de posibilidades para incorporar sensores para la róbotica, incluso pueden probarse protocolos de comunicación no estándares y que sean hechos a las necesidades del proyecto.

Una vez importado el módulo hardware que opera bajo las reglas del bus I2C, se incorpora fácilmente al sistema y hacer que el procesador Microblaze lo vea como un periférico más. Esto se logra mediante utilidades que proporciona la herramienta EDK.

Dado que el robot Create solamente cuenta con un mecanismo de codificadores rotatorios para calcular el giro, éste en algunas ocasiones puede no ser exacto, por distintas razones: tipo de superficie, resbalones en las llantas, llantas desgastadas, líquidos derramados, etc. De esta forma podemos hacer uso de sensores extras para mejorar este tipo de fallas. Si bien la brújula no garantiza un giro exacto del robot si mejora en gran medida el desempeño de esta tarea.

En la tabla 5.1 se observan los resultados obtenidos al incorporar la brújula para realizar la corrección de giro, realizando la comparación entre usar y no usar la brújula. Se realizaron 20 giros por cada uno de los grados que se enlistan en la tabla y lo que se muestra es el promedio de error de las 20 lecturas.

Giro (grados)	Error (grados)	
	Sin brújula	Con brújula
45	3.5	1.2
90	4.2	1.0
135	5.4	1.5
180	6.7	1.5
225	5.8	1.2
270	6.6	1.3
315	6.2	1.4
360	5.5	1.0
60	3.3	1.2
120	4.4	1.3

Tabla 5.1: Resultados para la prueba de corrección de giro, en donde se muestra la utilidad del uso de una brújula para corregir el giro de un robot

La desventaja de usar la brújula como corrector de giro es que depende de los elementos que haya en el ambiente de trabajo, elementos metálicos o magnetizados pueden influir en la lectura de la brújula.

Lo realmente notable de la prueba es que en los dispositivos FPGA se pueden añadir a la plataforma múltiples tipos de sensores con sus respectivos protocolos para lograr un mejor desempeño. En este experimento se realizó la prueba para un sensor I2C pero pueden utilizarse otros protocolos de comunicación que vienen incluidos en EDK (e.g. I2C, SPI, CAN, Ethernet, USB, etc.), o bien buscar códigos fuente gratuitos (o pagar licencia para algunos), o bien realizarlo de forma personalizada y una vez que se tiene el código para el protocolo de comunicación se puede importar al sistema Microblaze mediante la utilidad *"emphCreate/Import Peripheral"* que ofrece la herramienta EDK, la cual hace que el proceso de incorporar el perférico al sistema sea transparente para el usuario.

5.3.2. Hardware en paralelo: Sonares

Esta prueba consiste en mostrar la diferencia de implementar tareas en software contra la implementación en hardware. Un ejemplo aplicado a la robótica móvil es el uso de sonares, cuando se adquieren este tipo de dispositivos generalmente entregan ya sea un ancho de pulso en donde se deja la tarea de interpretar la distancia al usuario o bien entregan la distancia en cm pero de forma secuencial. La ventaja de este última forma es que los datos se dan en una medida estándar (e.g. en milímetros, centímetros, etc.) y utiliza un sólo pin de comunicación de datos, la desventaja es que la tarjeta controladora debe tener el protocolo de comunicación implementado para poder leer los datos. Por otro lado, utilizar un sonar que entregue un ancho de pulso ofrece flexibilidad, ya que el usuario decide cual va a ser la forma de comunicación hacia la tarjeta controladora y decide si se hace de forma secuencial o en paralelo. Entregar un ancho de pulso como salida no está sujeto a utilizar cierto tipo de comunicación, aunque la desventaja es que se debe que realizar la tarea de conversión del ancho de pulso hacia una medida estándar, pero esta desventaja se puede aminorar si se utilizan dispositivos FPGAs, como se describe a continuación.

Hoy en día se utilizan sonares que ofrecen una salida convertida en milímetros, lo cual es bastante útil, pero también se siguen usando sonares que ofrecen como salida solamente un ancho de pulso (e.g. SRF05 (Acroname, Diciembre 2008) y PING ultrasonic (Parallax, Septiembre 2008)), la elección de adquirir este tipo de sonares y no los que ya ofrecen una salida estándar, puede ser por el precio y rango de operación. De esta forma, en una tarjeta controladora como la Handy Board, si se decide utilizar un sonar en su forma de ancho de pulso, usualmente la tarea de

		Software		Hardware	
Distancia al objeto (cm)	Número de sonares	ciclos de reloj	tiempo (ms)	ciclos de reloj	tiempo (ms)
10	1	70,300	1.4	45,600	1.1
	4	274,450	5.48		
20	1	98,650	1.97	69,870	1.4
	4	386,050	7.72		
35	1	140,100	2.8	110,000	2.2
	4	564,700	11.3		
60	1	212,650	4.25	195,100	3.9
	4	854,350	17.0		
100	1	328,500	6.57	305,210	6.1
	4	1,323,000	26.4		
150	1	475,200	9.5	440,050	8.8
	4	1,943,700	38.8		

Tabla 5.2: Resultados para la prueba de sonares, la conversión software muestra ser más lenta que la conversión en hardware y se observa que los tiempos en hardware son constantes si se usan uno o los cuatro sonares.

convertir el ancho de pulso en una distancia estándar (e.g. milímetros) es una tarea que se le asigna a una función en software, en donde se tiene que esperar hasta que la tarea de conversión sea terminada para poder continuar con la ejecución del programa. Más aún si el robot tiene un anillo de sonares, el tiempo que se tarda la conversión de todos los sonares se puede incrementar y afectar la velocidad de respuesta del robot.

De esta forma, usando el FPGA se implementa la tarea de conversión en hardware, activando todos los sonares al mismo tiempo y esperando solamente una "bandera" en hardware que indica que la conversión esta lista, de esta forma, la única tarea en software es activar los sonares y leer el resultado convertido en centímetros.

Así se realiza una aplicación en donde se compara el tiempo que se tarda la lectura de uno y cuatro sonares tanto en software como en hardware, los resultados obtenidos se observan en la tabla 5.2. Los sonares usados para esta prueba fueron cuatro sonares *PING ultrasonic* (Parallax, Septiembre 2008).

Con esta prueba se observa la ventaja de implementar módulos hardware en los dispositivos FPGA, es decir, se pueden realizar módulos que trabajen en paralelo y que de alguna manera ayuden al procesador a realizar otras tareas mientras los módulos hardware se encuentran trabajando. Además se pueden añadir más de un sonar que algunas tarjetas que no tienen los recursos suficientes para hacerlo (e.g. Handy Board que sólo da soporte a un sonar). En la tarjeta FPGA usada para este trabajo pueden añadirse aproximadamente hasta diez sonares, esto depende de que otros sensores se conectan al puerto de expansión.

Esta idea puede extenderse a otras aplicaciones, por ejemplo, en hardware se pueden estar realizando tareas de procesamiento de imágenes, procesamiento de señales, etc., mientras el procesador se dedica a otras cosas, mejorando así la velocidad de respuesta del sistema.

5.3.3. Co-procesador para imágenes

Esta prueba consiste en realizar un co-procesador para imágenes el cual tenga múltiples módulos de convolución trabajando en paralelo y acelerando el tiempo de procesamiento. La prueba es comparar la implementación software contra la implementación en hardware. La metodología seguida para realizar esta prueba es la siguiente:

1. Importar el co-procesador para imágenes hecho en XSG al sistema Microblaze en EDK y

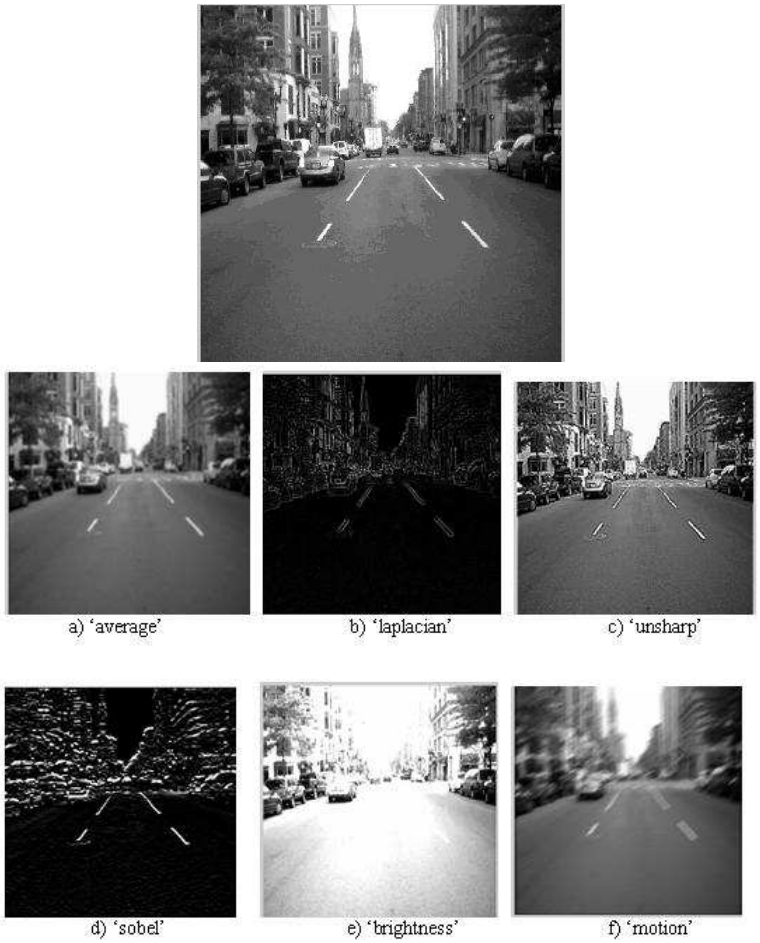


Figura 5.13: Ejemplos de tipos de filtros que se pueden aplicar usando el co-procesador para imágenes. El usuario puede configurar el tamaño de la máscara, los coeficientes de la máscara y el formato en cómo se representan estos coeficientes. Una descripción detallada de este tipo de filtros puede ser encontrada en González et al. (2004); Escalera (2001)

realizar las interconexiones necesarias con el bus FSL.

2. Desde una PC con MATLAB©, pasar la imagen a la memoria SDRAM de la tarjeta por medio del puerto serie.

3. Realizar el procesamiento de la imagen (a nivel software o hardware) y guardar la imagen resultante en la SDRAM.

4. Pasar la imagen resultante hacia la PC con MATLAB©por medio del puerto serie.

5. En MATLAB©realizar post-procesamiento para comparación de imágenes, medir desempeño, etc.

En la figura 5.13 se observan algunos de los posibles tipos de filtros que pueden ser aplicados a la imagen original, en realidad se pueden implementar una buena variedad de tipos de filtro con una representación en punto fijo de los coeficientes de la máscara, estos dos parámetros son configurables por el usuario.

Tamaño de máscara	Formato coeficientes	Coeficiente de correlación D
3×3	UFix_8_8	1.00
3×3	UFix_8_6	1.00
3×3	UFix_8_4	0.09
3×3	UFix_8_8	0.98
3×3	UFix_8_6	0.14
3×3	UFix_8_4	0.05

Tabla 5.3: Comparación de evaluar el proceso de convolución entre las imágenes software vs hardware. El coeficiente de correlación D muestra que tan parecidas son las imágenes hardware y software

La primera prueba realizada es para verificar qué tan confiables son los resultados arrojados por la parte hardware, así que se realiza una comparación entre los resultados a nivel software (mediante la herramienta MATLAB©) y los resultados a nivel hardware (usando el co-procesador para imágenes).

La prueba consiste en realizar el proceso de convolución a una imagen de tamaño 200×200 y máscaras de tamaño 3×3 y 5×5, con un filtro del tipo gaussiano con los coeficientes mostrados en la figura 5.14

0.1019	0.1154	0.1019
0.1154	0.1308	0.1154
0.1019	0.1154	0.1019

0.0232	0.0338	0.0383	0.0338	0.0232
0.0338	0.0492	0.0558	0.0492	0.0338
0.0383	0.0558	0.0632	0.0558	0.0383
0.0338	0.0492	0.0558	0.0492	0.0338
0.0232	0.0338	0.0383	0.0338	0.0232

Figura 5.14: Coeficientes para el filtro gaussiano. Se muestran los coeficientes para una máscara de 3×3 (arriba) y para una máscara de 5×5 (abajo)

Posteriormente se comparan los resultados del co-procesador contra los arrojados por el *toolbox* de Imágenes de MATLAB©. Después se obtiene una medida de similitud entre la parte hardware y la software, dicha medida de similitud llamada Coeficiente de Correlación D, esta dada por la ecuación 5.1 y obtiene el cociente entre el número de pixels que son iguales en ambas imágenes entre el número de pixeles diferentes.

$$D = num\,pixels\,iguales\,/\,num\,pixels\,totales \tag{5.1}$$

La tabla 5.3 muestra estos resultados. De los resultados obtenidos se puede inferir que el formato[4] con el que se representan los coeficientes de la máscara es muy importante; pero en general un formato con 10 bits para representar la parte decimal es suficiente para obtener buenos resultados en el caso de tener coeficientes con parte fraccional.

[4]El formato para los coeficientes viene dado por tres campos, separados por un guión bajo: el primero puede ser Fix o UFix indicando si el número representado es signado o no signado respectivamente, el segundo campo es el número de bits de todo el número y el tercer campo representa el número de bits destinados a la parte fraccional. Ejemplo: UFix_8_4 (Sin signo de 8 bits, de los cuales 4 bits son para la parte fraccional)

Tamaño de la imagen	Tipo de filtro	Tiempo Software (seg)	Tiempo Hardware (seg)	Coeficiente de correlación	Formato de coeficientes
$200x200$	Gaussiano ($3x3$)	4.4	0.55	0.96	UFix_8_8
	Gaussiano ($9x9$)	6.4	0.88	0.94	UFix_10_10
	Unsharp ($3x3$)	3.3	0.46	0.99	Fix_10_6
	Laplaciano ($3x3$)	2.1	0.29	0.98	Fix_10_7
	Sobel ($3x3$)	1.9	0.16	1	UFix_10_10
$300x500$	Gaussiano ($3x3$)	7.5	0.81	0.96	UFix_8_8
	Gaussiano ($9x9$)	10.9	1.20	0.94	UFix_10_10
	Unsharp ($3x3$)	6.3	0.69	0.99	Fix_10_6
	Laplaciano ($3x3$)	4.3	0.55	0.98	Fix_10_7
	Sobel ($3x3$)	3.9	0.36	1	UFix_10_10
$480x640$	Gaussiano ($3x3$)	9.1	1.19	0.96	UFix_8_8
	Gaussiano ($9x9$)	15.2	1.74	0.94	UFix_10_10
	Unsharp ($3x3$)	8.1	0.86	0.99	Fix_10_6
	Laplaciano ($3x3$)	6.2	0.80	0.98	Fix_10_7
	Sobel ($3x3$)	5.8	0.73	1	UFix_10_10

Tabla 5.4: Resultados para la prueba del co-procesador de imágenes. Los resultados muestras como la implementación hardware acelera significativamente los resultados sobre la implementación en software

Como segunda prueba se tiene la comparativa en tiempo de la operación de convolución entre la implementación software contra la implementación hardware. Por lo que se refiere a la implementación software se implemento la convolución realizando el algoritmo en software usando el procesador Microblaze, mientras que para la implementación hardware el proceso de convolución se realiza mediante el co-procesador y también usando el procesador Microblaze. En la tabla 5.4 se observan los resultados obtenidos, en esta tabla también se observa el coeficiente de correlación D.

Con esta prueba se puede observar nuevamente la capacidad del poder del paralelismo que tienen los dispositivos FPGA y de como este paralelismo puede ayudar a lograr mejores desempeños en diversas tareas relacionadas con el procesamiento de datos para aplicaciones de Robótica Móvil.

5.3.4. Pruebas generales: Seguidor de pared y laberinto

Este experimento consiste de dos pruebas que tienen la finalidad de comprobar el correcto funcionamiento de la plataforma FPGA, es decir, que todo en conjunto funcione (plataforma FPGA junto con robot Create, brazo manipulador, sonares, infrarrojos y brújula).

La primera prueba es un seguidor de pared, la finalidad del robot es llegar de un extremo del área de prueba a otro, permaneciendo a cierta distancia (10 cm aproximadamente) de la pared. Además el robot debe transportar objetos desde la posición inicial hasta la posición final, haciendo uso del brazo manipulador. En esta prueba se hace uso del robot Create, de los cinco sonares para determinar la distancia de la pared y determinar en que momento debe girar, se usa también la brújula para la corrección de giro y por último se hace uso del brazo manipulador. Esta primera prueba presentó algunas dificultades para que el robot siguiera la pared, pero dichas dificultades se vieron resueltas al modificar el algoritmo y las condiciones de respuesta de los sonares. En cuanto a la parte hardware se presentaron dificultades del lado de los cables de conexión para los sonares y con el brazo manipulador, se realizaron algunos ajustes de cables y los sensores funcionaron correctamente. La prueba se realizó veinte veces, de las cuales las últimas diez cumplieron con el objetivo de la prueba y el comportamiento del robot fue estable

(i.e. llegó a la posición final manteniendóse a una distandia de 10 cm de la pared).

La segunda prueba consta de encontrar la salida en un laberinto, desde una posición inicial se comienza la búsqueda de la meta a través de un laberinto de líneas, estas líneas fueron "dibujadas" con cinta adhesiva color negra. Para esta prueba los elementos que entran se usan son: el robot Create, sensor IR seguidor de línea y brújula. Nuevamente las dificultades fueron más aspectos externos a los objetivos del trabajo (e.g. sensor infrarrojo flojo y que se caía del robot, brújula que no se encontraba bien sujeta a la base del robot, etc.). La prueba se realizó veinte veces al principio para tipo de laberinto, siendo las últimas cinco veces con resultados satisfactorios (i.e. el robot encontraba la salida y permanecía sobre las líneas del laberinto). Después se probaron otros dos tipos de laberintos (i.e. con formas distintas) y la prueba se realizó ocho veces para cada laberinto, siendo las últimas cinco la que mostraba resultados satisfactorios. En la figura 5.15 se muestra el diagrama general de estas dos pruebas.

Con este tipo de pruebas se demuestra la flexibilidad de los dispositivos FPGAs, de poder integrar al robot una buena variedad de sensores/actuadores para robótica móvil, siempre y cuando sean compatibles con la plataforma FPGA. Además de poder implementar diferentes tipos de protocolos de comunicación, incluso probar protocolos nuevos o protocolos hechos a las necesidades del proyecto, pueden implementarse aceleradores en hardware que ayuden a paralelizar procesos, entre otros usos. Por otro lado, la capacidad de reconfiguración de los FPGAs, permiten que usando el mismo dispositivo puedan implementarse plataformas diferentes, para robots, sensores y actuadores totalmente distintos, o bien añadir o eliminar módulos hardware dependiendo de lo que se quiera usar de la plataforma. Más aún esto último puede realizarse de forma dinámica, es decir, en "tiempo de ejecución" del robot al vuelo ir reconfigurando el hardware dependiendo de la respuesta de los sensores.

5.4. Discusión

En las secciones anteriores se habló de las ventajas que se observaron en los experimentos realizados. Pero si bien es cierto que los FPGAs tienen varias virtudes, también existen desventajas en estos dispositivos. En la tabla 5.5 se presenta una tabla comparativa de las ventajas y desventajas que conllevan a usar dispositivos FPGA como plataformas robóticas.

Ventajas	**Desventajas**
• Flexibilidad para implementar lógica digital • Capacidad de Reconfiguración • Paralelismo a nivel Hardware y Software •Opción de implementar los recursos necesarios que demanda la aplicación (compromiso Velocidad vs Área) • Gran cantidad de IP *cores* disponibles y optimizados para el usuario • Buena cantidad de recursos E/S • Consumo de potencia con respecto a los procesadores comerciales • Posibilidad de implementar *custom* IP *cores* y añadirlos al sistema empotrados • Capacidad de montar distintos SO de fabricantes *third party* • El procesador puede migrarse a distintas familias de FPGAs • Capacidad de implementar sistemas multi-procesadores en un solo chip	• Necesidad de añadir e implementar los módulos hardware (e.g. protocolos, lógica de usuario, etc.) • Complejidad para diseñar la plataforma • Dificultad para cambiar la funcionalidad, se requieren personas especializadas • Dependencia del sistema con las herramientas de diseño • Consumo de potencia mayor con respecto a los microcontroladores (pero menor que las PCs) • Costo del dispositivo y elementos de la tarjeta FPGA • Esfuerzo en implementar los módulos hardware

Tabla 5.5: Tabla comparativa de las ventajas y desventajas de usar FPGAs en plataformas robóticas

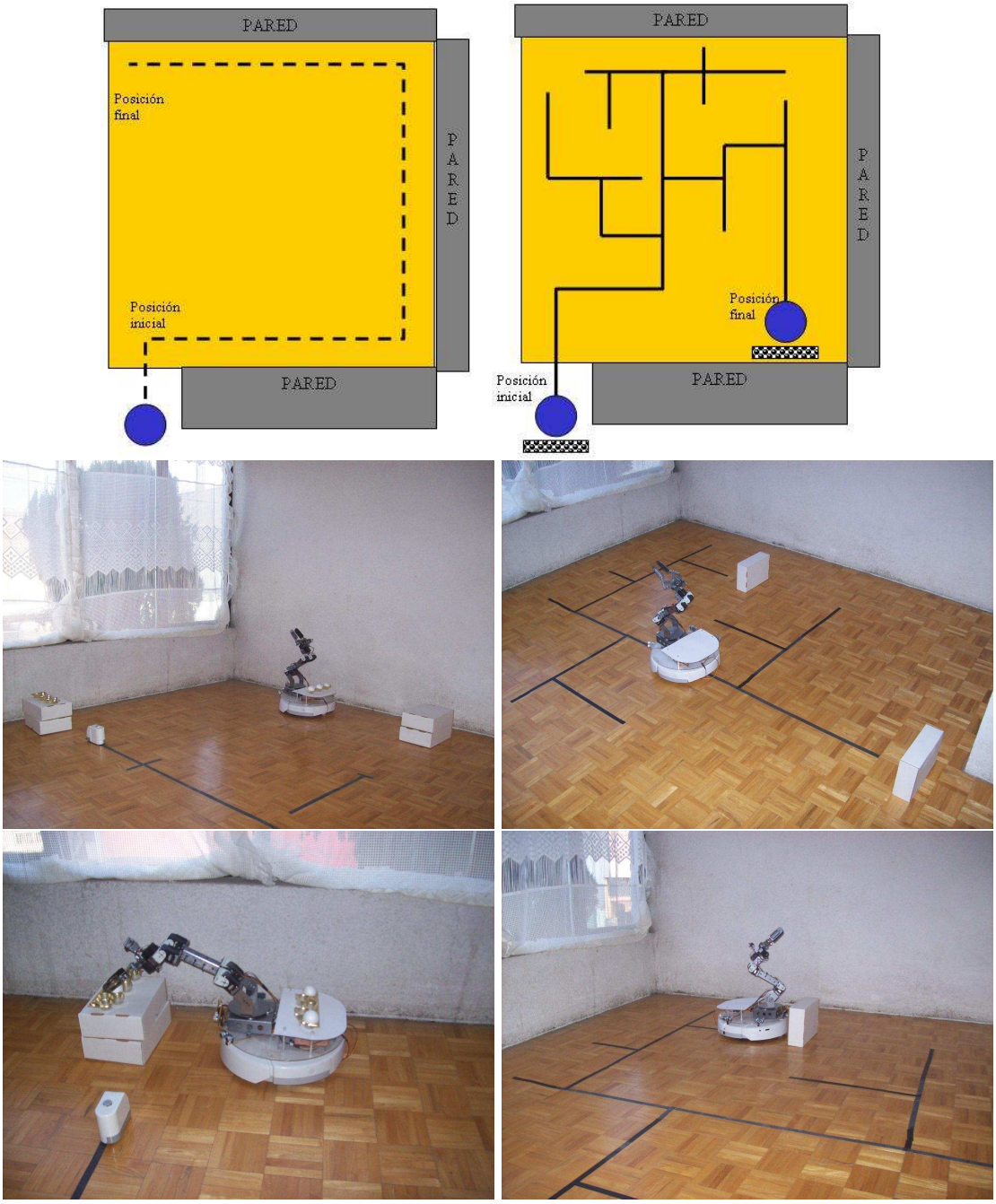


Figura 5.15: Esquema y fotos de las pruebas generales para el robot de prueba. La prueba del seguidor de pared (columna izquierda) y la prueba de transporte de objetos (columna derecha)

Los puntos listados en la tabla 5.5 son en una perspectiva general del uso de los FPGAs en plataformas robóticas. en la tabla 5.6 se compara específicamente la plataforma FPGA propuesta en este trabajo con otras plataformas comerciales, la elección de estas plataformas se basaron en las que se usan más en el mercado y las más populares entre la comunidad de robótica. En la tabla 5.6 también se destaca la mejor plataforma para cada característica.

Característica	**Handy Board** uC	**Hanndy Cricket** uC	**Bot Board II + procesador** uC	**KoreBot** uP	**Gumstix** uP	**EyeBot** uC	**CMUcam3** uP	**Plataforma FPGA**	**Mejor**
Procesador	Motorola 8bit 68HC11 2 MHz	Microchip 8bit PIC16C715 4MHz	BasicAtom-Pro16bit H8/3664F 100 K IPS	Intel 32 bit XSCALE PXA-255 400MHz	Verdexpro Marvell PXA270 400-600 MHz	Motorola 68332 32bit Controller 25 MHz	NXP LPC2106 ARM7 60 MHz	Microblaze 32bit 105 MHz	Gumstix
Potencia comp. estimada	0.5 - 1 MIPS	1 - 2 MIPS	0.1 MIPS	200 - 300 MIPS	200 - 500 MIPS	15 - 20 MIPS	40 - 50 MIPS	1-10 GOPS	plat FPGA
Software	Interactive C Java	Cricket Logo	AtomPro Language	Linux 2.4.19	Linux 2.6.27	RoBIOS	Custom C code	Petalinux (Linux 2.6.20)	KoreBot, Gumstix y plat FPGA
Memoria	32 kB	4 kB	4 kB	64 MB	64 MB	1 MB	64 KB	64 MB	KoreBot, Gumstix y plat FPGA
Características y soporte para Sensores/Actuadores	- 4 motores DC (1 A) - pantalla LCD - 7 entradas analógicas - 9 entradas digitales - 2 IR (trans/recep) - Bocina	- 4 motores DC - 1 sensores IR -6 sensores digitales - 8 servos - bocina	-PlayStation Conector - Leds,botones - 16 pines E/S - 16 bus (Servos, motores DC, pines E/S) - Socket 24/28 pines - Bocina	- 1KB-250 Interfaz (3 Seriales, 4 USB, 1 I2C, 1 SPI, 1 AC97, 53 GPIOs) - 2 RS232 - 1 USB Cliente	- USB host - 3 UARTs - hasta 90 GPIOs - 1 I2C Bus - 1 Bluetooth	- camera digital - 2 motores DC con encoders - 12 servos -6 sensores IR (o 6 entradas digitales) - 2 parachoques (o 2 entradas digitales) - 6 entradas analógicas - LCD - bocina - micrófono	- sensor RGB color 352x288 - 4 servos	- Sensores IR - Brazo Manipulador - Sonares - Brújula	KoreBot y plat FPGA
Expansión	- conector SPI - conector analógico - Tarjeta de Expansión (sensores analógicos, LEGO, salidas digitales, servos)	- Bus Port (tarjetas de display, motores DC, servos, reles, leds, etc)	- SSC-32 - SaberTooth	- KoreMotor (4 motores DC) - KoreConnect (RS232 y USB) - KoreSound (audio E/S) - KoreIO (pines E/S analógicos, salidas digitales, servos)	-RoboAudio -Robostix (SPI, I2C) -GPSstix (GPS,Audio,LCD,USB)	- EyeCam (sensor CMOS) 640x480 pixels	–	- Añadir más tarjetas a través de los puertos de expansión - hasta 70 pines E/S	plat FPGA

Característica	**HandyBoard** uC	**Handy Cricket** uC	**Bot Board II** + **processors** uC	**KoreBot** uP	**Gumstix** uP	**EyeBot** uC	**CMUcam3** uP	**Plataforma FPGA**	**Mejor**
Protocolos	- SPI - UART	–	- UART	- I2C - UART - SPI - USB	- I2C - UART - USB - Bluetooth	- UART	- UART - SPI	- UART - UART 16550 - I2C - SPI - CAN - Ethernet - Inalámbrico	plat FPGA
Precio (USD)	$ 299	$ 139	$ 25 + $60 processsor	$ 440 + $190 exp.	sola $159 + ($30 - $130)	$ 1,000	$ 239	$ 490 ($ 250 *custom*)	Bot Board
Tamaño (cm)	10.8 x 8	5.7 x 4.8	10 x 8	8.5 x 5.7	8 x 2	10.6 x 10	5.5 x 5.7	15x17 (10x12 *custom*)	Gumstix
Versatilidad (agregar + periféricos)	baja	baja	baja	mediana	mediana	baja	baja	alta	plat FPGA
Procesamiento paralelo	Si (SW)	No	No	Si (SW)	Si (SW)	No	No	Si (HW y SW)	plat FPGA
Peso (grs)	200 aprox.	<200	90 aprox	35	8	115 aprox	200 aprox	250 (120 *custom*)	Gumstix
Alimentación	12 v DC, 500 mA	5 - 6 v DC, 500 mA	9 - 5 v DC, 500 - 250 mA	5 v DC, 250 mA	3.6V - 5.0V DC, 500 mA	7.2 - 9 v DC, 270 mA	5v DC, 130 mA	5 v DC, 500m - 1A	CMUcam3 y KoreBot
Notas	Buena para el control de algunos motores y sensores. No puede hacer tareas de visión ni procesamiento intensivo de datos	Es una versión menos potente que la HandyBoard pero más ligera, pequeña y barata	Para controlar diversos tipos de robots pero con tarjetas extras y de la misma compañía	Buen desempeño y recursos de expansión. Buenas tarjetas de expansión. Tarjetas de la misma compañía	Mini-computadora de propósito general. Las tarjetas de expansión sólo se puede tener una a la vez	Se utiliza sólo como tarjeta que se añade a un robot. Acepta un tipo de cámara. Mediana resolución	Se utiliza como tarjeta que se añade a un sistema embebido para realizar tareas de visión	Implementar diferentes plataformas hechas a la medida Necesidad de personas especializadas	–

Tabla 5.6: Comparativa entre la plataforma FPGA y las plataformas comerciales. Datos tomados a la fecha de Diciembre-2008. Excepto por las propiedades 'Peso' y 'Alimentación' tomadas a la fecha de Febrero-2009. El criterio de evaluación para la característica de 'Versatilidad' se define en la discusión de las páginas siguientes

A continuación se realiza una discusión de los puntos de la tabla 5.6 acerca del por qué se seleccionaron a esas plataformas como la mejor en algunas de las principales características.

Con respecto al procesador y su desempeño, el procesador que ofrece el mejor desempeño es el de la tarjeta Gumstix y de la plataforma FPGA, aunque esta última corra sólo a 105 MHz. La diferencia está en que los FPGAs con el poder de paralelismo que tienen pueden lograr varias Giga Operaciones por Segundo (GOPS), dar una cantidad exacta sería algo complicado pero se puede dar un rango aproximado de 1-10 GOPS dependiendo de la aplicación y de las operaciones. Algunos trabajos relacionados acerca de arquitecturas FPGA que logran varias GOPS son (Strenski, 2007; Sánchez, 2008; Malachy, 2008; Kreindler, 2008; Rousseaux, 2008), con estos resultados los FPGAs estarían arriba de los procesadores actuales, que alcanzan los 2-3 GOPS. Si bien es cierto que la frecuencia de operación de varios de los procesadores vistos en la tabla 5.6 rebasan a la frecuencia de operación del Microblaze, con el uso de dispositivos FPGA y la capacidad de paralelismo se puede alcanzar varias GOPS en una determinada aplicación, en donde si esta aplicación se realiza en un procesador secuencial sería más lento.

En cuanto al software, la mejor opción es una plataforma que trabaje con un SO Linux, por dos principales razones. Un SO ofrece una capa de abstracción al hardware, ya que lo hace de forma transparente esto hace más fácil manipular los periféricos hardware. La otra razón es usar el SO basado en Linux por las ventajas que este tipo de SO ofrece, como es estable, confiable, mucho códigos fuente son abiertos y existe una gran comunidad de programadores, esto da un buen "soporte técnico". Es por esto que las plataformas FPGA, Gumstix y KoreBot son la mejor opción.

Por lo que respecta a la capacidad para manipular sensores/actuadores, se tiene que las plataformas FPGA y KoreBot son las mejores, ya que la plataforma KoreBot da soporte a una buena variedad de protocolos y además tiene una buena cantidad de pines E/S para poder conectar más sensores/actuadores digitales. La plataforma FPGA da soporte también a los sensores/actuadores más comúnmente utilizados en robots móviles (e.g. brazo manipulador, sonares, brújula, etc.), además tiene una buena cantidad de pines E/S y circuitería necesaria para añadir más de estos sensores. Las mayoría de las demás plataformas controlan motores DC, algunas servos, y protocolos específicos que algunas los tienen y otras no.

En cuanto a los protocolos que soportan, la mejor es la plataforma FPGA ya que es posible implementar la mayoría de los protocolos digitales que existen en robótica móvil, simplemente se tiene que saber el tipo de comunicación que utilizan los sensores/actuadores e implementar estas interfaces con la lógica del FPGA, tal y como se demuestra implementando el protocolo I2C para poder comunicar a los sensores SRF02 y CMPS03 con la plataforma FPGA propuesta. Las demás plataformas necesitan una tarjeta de expansión para poder adquirir ciertos protocolos de comunicación y algunas otras simplemente carecen de algunos de ellos.

La capacidad de expansión que tienen las plataformas es otra característica importante, nuevamente la plataforma FPGA es la mejor opción ya que el FPGA ofrece la flexibilidad necesaria para poder implementar nuevo hardware para conectar tarjetas y más sensores/actuadores, existen FPGAs que ofrecen más de 100 pines E/S, en donde la ventaja esta en que esos pines E/S pueden funcionar para conectar una tarjeta que controle servos y esos mismos pines podrían funcionar también para controlar más sonares o para conectar alguna cámara, gracias a que el FPGA es un dispositivo reconfigurable en hardware, así que las posibilidades de expansión son muchas.

Las plataformas Bot Board y Gumstix son las mejores opciones en cuanto al precio y tamaño. La Bot Board es la más barata pero necesita de la compra de procesadores extras además de que no ofrece muchas opciones de expansión. La Gumstix es la más pequeña en cuanto a tamaño, esto es una buena característica ya que puede incorporarse en robots muy pequeños (e.g. robots aéreos de dimensiones 3×10 cm). Aunque también se observa que la diferencia no es mucha, la plataforma FPGA aún así tiene un tamaño pequeño y es liviana para poderse incorporar en robots móviles pequeños (20×20 cm aprox.). Muchas de las plataformas son baratas pero el

desempeño y el soporte para sensores/actuadores se ve reflejado también. Quizá las plataformas que ofrecen un mejor balance entre precio-tamaño-desempeño son la plataforma FPGA, Gumstix y la KoreBot. La plataforma FPGA porque como ya se ha mencionado se puede realizar una tarjeta especialmente diseñada para soportar aplicaciones de robótica móvil en lugar de usar tarjetas FPGA comerciales, con esto se reduciría el tamaño y en cuanto al costo (10×12 cm aprox. y de $250 USD aprox.).

La tarjeta Gumstix es la más liviana y la diferencia con respecto a las demás tarjetas es significativa (8 grs vs 250 grs), esta característica puede ser de gran utilidad en robots pequeños y que necesiten una carga ligera (e.g. robots aéreos pequeños de dimensiones 3×10 cm). El resto de las tarjetas, incluyendo la tarjeta FPGA Spartan 3E 1600 ofrecen un peso aceptable para robots móviles de dimensiones de alrededor de 20×20 cm, en general todas las tarjetas tienen un peso liviano comparado con un robot que debe soportar una computadora portátil. Cabe señalar que si se realiza una tarjeta hecha a la medida como la que se muestra en la figura 4.1 el peso puede reducirse aproximadamente a la mitad (120 grs aprox.).

Para la alimentación las tarjetas con menor consumo de potencia son KoreBot y CMUcam3, las cuales son las que menos demanda corriente tienen. Cabe señalar que esto depende del tipo y del número de sensores que soportan las tarjetas, por ejemplo, la tarjeta CMUcam3 pide 130 mA pero solamente pueden funcionar la cámara y cuatro servos. Para una tarjeta que tenga más aditamentos como sonares, brújula, cámara, etc. el consumo de corriente será mayor.

La versatilidad para agregar nuevos periféricos está evaluada de la siguiente manera: una "poca" versatilidad, significa que puede dar soporte a menos de 3 protocolos de comunicación y que da soporte a sensores/actuadores básicos como motores DC, servos, sonares, infrarrojos, LEDs y botones. Una versatilidad "mediana" significa que cubre los requerimientos de la básica, que da soporte de tres a cinco protocolos de comunicación y además tiene un puerto de expansión con al menos 30 pines E/S. Mientras que una versatilidad "alta" significa que cubre los requerimientos de una versatilidad mediana, que tiene más de cinco protocolos de comunicación y además pueden añadirse módulos hardware hechos por el usuario, está última característica está más enfocada a dispositivos configurables en hardware.

De esta forma, por sus capacidades de expansión y la flexibilidad que tiene la plataforma FPGA para dar soporte a nuevas interfaces de comunicación, la plataforma FPGA presenta la mejor versatilidad para agregar nuevos periféricos. También los periféricos internos son importantes, como los módulos de conversión de sonares y los co-procesadores para imágenes que se presentaron en este trabajo, esto demuestra que se pueden implementar una gran variedad de periféricos internos/externos dependiendo de las necesidades del proyecto.

El procesamiento paralelo es una característica importante para mejorar la rapidez con que se realizan ciertas operaciones. La gran mayoría de las plataformas que existen comercialmente basan su procesamiento en microcontroladores o procesadores que ejecutan instrucciones de forma secuencial, la ventaja de la plataforma FPGA es que se pueden implementar co-procesadores, que se basan en módulos hardware trabajando en paralelo para ejecutar más rápidamente dichas tareas, es gracias a esta capacidad que los FPGAs pueden alcanzar las GOPS que se mencionaron.

También en la tabla 5.6 se tiene una comparativa contra tarjetas que se utilizan principalmente para tareas de visión. La visión es una de las capacidades sensoriales más importantes en un robot móvil. Estas plataformas son de las más utilizadas en el mercado y tienen un desempeño aceptable. Por otro lado con la plataforma FPGA propuesta se demostró que es posible y viable implementar arquitecturas y módulos hardware para tareas de visión. Se realizó un módulo coprocesador que realiza la tarea de convolución en hardware, una de las operaciones más utilizadas e importantes en este campo. Las plataformas EyeBot y CMUcam3 tienen la desventaja de trabajar con un solo tipo de cámara y de que no tienen muchas opciones de controlar sensores/actuadores, más bien se utilizan como aditamentos a otras tarjetas. De hecho se puede dar soporte para que estas dos tarjetas se puedan conectar y funcionar apropiadamente junto con la plataforma FPGA, en contraste no todas las plataformas comerciales pueden dar

soporte a la EyeBot o la CMUcam. También con la plataforma FPGA es posible conectar una gran variedad de sensores de imagen, nuevamente la interfaz puede ser implementada mediante la lógica del FPGA. Más aún, se pueden implementar arquitecturas especializadas para realizar tareas de visión y lograr una aceleración en el procesamiento de datos, algunos trabajos relacionados son (Arimoto, 2008; McBader & Lee, 2002).

En la actualidad existen sensores "inteligentes", es decir, sensores que ya tienen incorporados microprocesadores (o algún otro elemento hardware) que realizan el procesamiento de datos y que dan como resultado información que le es útil al usuario, ejemplo de este tipo de sensores con las cámaras CMUcam, las cuales realizan la tarea de procesamiento de imágenes a alto nivel. En este sentido, pareciera que la incorporación de co-procesadores dentro de los FPGAs es innecesaria, pero el FPGA permite la flexibilidad de incorporar tareas de co-procesamiento que no necesariamente se encuentren incluidas en el sensor o algunas otras tareas que sean computacionalmente intensivas (e.g. procesamiento de señales).

Con esto finaliza la sección de resultados y gracias a las pruebas realizadas se da un buen indicio de que los dispositivos FPGAs pueden ofrecer ventajas interesantes en las plataformas robóticas. Las pruebas realizadas en este capítulo muestran algunos resultados interesantes, pero la finalidad principal es dejar un buen panorama y mostrar la viabilidad para que la plataforma FPGA crezca y de soporte a arquitecturas más complejas que reflejen más fielmente las ventajas de estos dispositivos.

Capítulo 6

Conclusiones

6.1. Conclusiones

Las plataformas comerciales para robótica móvil están basadas su gran mayoría en microcontroladores, los cuales sirven para controlar un conjunto de sensores/actuadores y ponerlos en funcionamiento en un robot móvil. La falta de poder computacional (i.e. procesamiento intensivo de datos) en los microcontroladores hace que este tipo de plataformas sirvan solo para el control de los sensores/actuadores sin esperar mucho acerca del procesamiento de datos que puedan realizar para algunas aplicaciones (e.g. visión por computadora). Otro aspecto de las plataformas basadas en microcontroladores o microprocesadores es que a pesar que cuentan con tarjetas de expansión y recursos para conectar varios tipos de sensores/actuadores, estos recursos siguen siendo limitados y hacen que el usuario o grupos de investigación tengan más de una plataforma comercial para sus proyectos. La plataforma FPGA propuesta es una solución que ofrece recursos para realizar aplicaciones que requieran procesamiento intensivo de datos, además de dar soporte a una gran variedad de sensores/actuadores como sonares, brújula, brazo manipulador, etc. La capaciadad de reconfiguración en hardware que tienen los FPGAs los convierten en una plataforma que puede ser flexible para implementar diversos tipos de plataformas con arquitecturas y elementos hardware hechos a las necesidades del proyecto.

La plataforma FPGA esta basada en un sistema empotrado con un procesador *soft* Microblaze y fue implementada sobre una tarjeta comercial que tiene un FPGA de la familia Spartan 3E usando las herramientas EDK, ISE y System Generator de Xilinx. Estas herramientas hacen varios procesos transparentes (e.g. conexión de buses, realización de coprocesadores, creación de lógica de usuario, etc.) al usuario facilitando la tarea de realizar el sistema empotrado en el FPGA. Se realizaron diferentes experimentos que tienen como objetivo principal dejar bases claras que apunten hacia mostrar que las plataformas FPGA pueden ser una buena opción para robots móviles por sus capacidades de reconfiguración hardware, flexibilidad y paralelismo.

La plataforma FPGA realizada para este trabajo presentó resultados satisfactorios en cuanto a su funcionamiento y desempeño en los experimentos realizados. Si bien la plataforma FPGA presentada en este trabajo no ofrece ventajas significativas aparentes con respecto a otras plataformas comerciales, se deja en claro el potencial de la plataforma FPGA para incrementarse y poder implementar arquitecturas que puedan controlar diferentes tipo de robots. Los experimentos realizados muestran algunas mejoras que se logran al implementar ciertas tareas en hardware (e.g. conversión de distancia en sonares, implementación de protocolos, co-procesadores, etc.). Pero el objetivo principal de estos experimentos fue dejar una buena base que indique que la plataforma FPGA puede extenderse y servir para diversas plataformas robóticas ofreciendo un buen desempeño.

Los dispositivos FPGA representan una solución flexible para la implementación de plataformas robóticas, ya que integra en un mismo chip gran variedad de IP *cores* ya sean ofrecidos por fabricantes o bien desarrollados por el usuario. Esto permite probar e implementar una gran variedad de sistemas con diferentes arquitecturas hardware, protocolos de comunicación, interconexión de componentes, etc.. La plataforma FPGA es flexible también a nivel software, ya que existe una buena variedad de soluciones para probar diferentes tipos de sistemas operativos de acuerdo a las características que se requieran.

Por otro lado, la plataforma FPGA es escalable también, ya que posee los recursos suficientes para añadir más dispositivos, sensores, actuadores, protocolos de comunicación, procesadores, etc., sin tener que modificar el sistema a nivel hardware, ya que basta con reconfigurar el FPGA. A nivel software la plataforma FPGA ofrece toda la infraestructura a bajo nivel para poder implementar una arquitectura software más elaborada, y que ésta pueda ampliarse para realizar funciones cada vez más complejas, nuevamente sin tener que rediseñar toda la infraestructura software que ya se tiene. Al principio se seleccionó un robot móvil para ser controlado mediante el FPGA, después se tomó la decisión de agregarle nuevos sensores/actuadores como el brazo manipulador y los sonares. Estos nuevos dispositivos se eligieron simplemente por ser una buena opción, es decir, no importó ni el fabricante ni si eran compatibles con la plataforma FPGA, esto demuestra que se pueden mezclar diferentes tipos de sensores/actuadores junto con robots móviles comúnmente usados en el mercado y que pueden ser controlados mediante la plataforma FPGA, es por ello que la plataforma FPGA es flexible y escalable. La plataforma FPGA está abierta para utilizar robots móviles que puedan ser controlados ya sea por un protocolo serial a bajo nivel o bien controlando directamente los motores, también es viable que la plataforma FPGA de soporte a otros sensores (e.g. cámaras, sensor de temperatura/humedad, acelerómetro) y actuadores (e.g. brazos manipuladores) comúnmente usados para robótica móvil.

Más en concreto, los FPGA permiten añadir a la plataforma interfaces de comunicación como las disponibles actualmetne para la mayoría de sensores en robótica (e.g. I2C, SPI, UART), ya que el protocolo puede ser implementado a nivel hardware mediante la lógica del FPGA, a diferencia de los microcontroladores, en los cuales se deben buscar aquellos que tengan los protocolos de comunicación a utilizar. Un microcontrolador es una arquitectura cerrada (i.e. inalterable). Por ejemplo, si un microcontrolador no tiene el protocolo I2C, sería complicado poder añadir esta característica, en donde se tendrían que buscar soluciones hardware como tarjetas de expansión que si tengan ese protocolo, o bien soluciones software en donde se tenga que programar este protocolo.

Una de las principales ventajas de usar dispositivos FPGA es su capacidad de reconfiguración, la cual permite que usando el mismo dispositivo FPGA puedan implementarse arquitecturas hardware diferentes, para robots, sensores y actuadores totalmente distintos, o bien añadir o eliminar módulos hardware dependiendo de lo que se quiera usar de la plataforma. En resumen, el FPGA permite integrar o eliminar hardware dentro del mismo dispositivo, esta característica es una gran ventaja con respecto a las demás plataformas. Como trabajo futuro, la reconfiguración también puede ser interesante ya que puede realizarse de forma dinámica, es decir, en "tiempo de ejecución" poder "re-programar" el hardware del FPGA dependiendo de la respuesta de los sensores.

Otra característica importante es el paralelismo inherente en los dispositivos FPGA, que permite acelerar procesos y ayudar a que el procesador pueda realizar otras tareas mientras los módulos hardware están trabajando. Este paralelismo permite que sea factible implementar módulos hardware que funcionan como co-procesadores, los cuales ayudan a mejorar el desempeño de la plataforma robótica, acelerando el tiempo de ejecución de las tareas. La programabilidad

de la arquitectura hardware mediante el uso de herramientas software (i.e. EDK) reduce en gran medida el tiempo de desarrollo del sistema, en donde varios de los procesos (e.g. importar un módulo hardware, añadir *cores*, conexión de *buses*, generación de controladores, etc) son transparentes al usuario. Aunque también se requiere invertir tiempo en aprender a usar este tipo de herramientas. Por el lado del software, usar Linux embebido como SO es una buena opción por las características ya conocidas de este SO, el cual proporciona confiabilidad, eficiencia y estabilidad.

La principal desventaja que presenta la plataforma FPGA es que para realizar una aplicación de usuario nueva, se necesita recompilar la imagen del SO Linux y volver a bajar esa imagen al FPGA, aunque no es algo que requiera mucho tiempo, aproximadamente 30 segundos. Lo mas conveniente es que el usuario programe todas las aplicaciones que necesite correr y bajar una sola vez esa imagen al FPGA. Por otro lado, si la plataforma FPGA se requiere cambiar a nivel hardware (i.e. agregar/eliminar un periférico, un protocolo, un coprocesador, etc.), esto no es trivial y es necesario contar con una persona especializada que sepa utilizar las herramientas de desarrollo (i.e. EDK y VHDL). Si bien esto es una desventaja que presenta la plataforma FPGA, es viable realizar algunas modificaciones para que la interfaz con el usuario sea más práctica y fácil de usar.

6.2. Trabajos futuros

Como trabajo futuro se propone explorar los siguientes aspectos:

- Explorar sistemas multi-procesadores que trabajen en el mismo FPGA con la intención de mejorar el desempeño total del sistema, distribuyendo las tareas de procesamiento entre los diferentes procesadores.
- Integrar una cámara al sistema y realizar *cores* para visión aprovechando el poder de procesamiento que poseen los FPGAs y desarrollar aplicaciones de navegación utilizando este tipo de dispositivo.
- Probar otro tipo de procesador, como el PowePC, que se encuentra en algunas de los dispositivos de la familia Virtex. Debido a que este tipo de procesador ofrece un mejor desempeño comparado con el Microblaze. Además evaluar otro tipo de SO embebidos, seleccionando el que mejor se adecúe a las necesidades del proyecto.
- Realizar pruebas con otro tipo de robots (e.g. Pioneer, PeopleBot, Kephera), otro tipo de sensores y actuadores (e.g. cámaras, acelerómetros, sensores de temperatura/humedad, GPS) para realizar una plataforma robótica con una mayor funcionalidad. Explorando así, protocolos de comunicación, arquitecturas hardware y elementos de lógica digital.
- Explorar alternativas de visualización disponibles para que el usuario pueda interactuar con la plataforma, como por ejemplo pantallas *touchscreen*.
- Probar *cores* que funcionen bajo el concepto de reconfiguración parcial, ya que esto puede ofrecer interesantes alternativas para aplicaciones en robótica móvil
- Crear una tarjeta FPGA con elementos hardware necesarios para poder dar soporte a una buena variedad de robots, sensores y actuadores.

Apéndice A

Implementación en EDK

A.1. Desarrollo del hardware del sistema

Varios de los IP *cores* utilizados para la plataforma FPGA ya están implementados y vienen como parte de la herramienta EDK de Xilinx. Algunos vienen con licencia gratuita y otros se adquieren pagando. Aunque también está la posibilidad de que el usuario realice su propio *core* desde código VHDL, hacerle un *wraping* para que funcione como un *black-box* e importar ese módulo al sistema embebido. Estos *cores* personalizados deben conectarse a los buses estándar de un sistema Microblaze o PowerPC como el bus OPB, PLB o bien conectarse como co-procesador usando el bus FSL. En este trabajo se hace uso de la técnica para añadir *cores* personalizados, y se describe en la sección A.3.

EDK es una herramienta que permite organizar e implementar sistemas embebidos completos, tanto la parte hardware como la software. El uso del EDK facilita en gran medida el diseño e implementación de sistemas embebidos, aunque su uso no es trivial, ya que se necesita cierto conocimiento en los sistemas embebidos, entendimiento de sus *cores*, manejo de memoria, utilizar los controladores o en caso de que no exista el controlador, realizar el *core* y después sus controladores para manipularlo, realizar las aplicaciones software para probar el hardware y cada uno de sus *cores*, etc.

En la figura A.1 se observa el entorno de trabajo de la herramienta EDK. La parte de *Project Information Area* (PIA) permite manejar información del proyecto, como visualizar archivos útiles como el MHS (*Microprocessor Hardware Specification*) que es el archivo en donde se especifican las características del Procesador, los periféricos y la dirección de memoria asociada a ellos, los buses y la conectividad en general de todo el sistema, se tiene también el archivo MSS (*Microprocessor Software Specifications*) el cual contiene la parte software del sistema, describiendo textualmente los elementos del sistema así como sus parámetros asociados a cada periférico. El PIA también gestiona la realización de proyectos software así como crear aplicaciones escritas por el usuario. Finalmente en este mismo panel se tiene el *IP Catalog*, que tiene una lista completa de todos los cores que EDK tiene disponibles para realizar un sistema empotrado, dentro de estos cores tenemos: procesadores (Microblaze y PowerPC), controladores de memoria, *buses*, UARTs, depuradores, relojes, temporizadores, *cores* hechos por el usuario, etc.

En el panel de *System Assembly View* (SAV) se tiene el *Bus Interface* en donde se interconectan todos los cores del sistema a los buses de manera general, mientras que en *Ports* se interconectan las señales E/S de cada core de forma detallada y finalmente en *Addresses* se asigna el espacio de memoria que cada core ocupa en el sistema empotrado.

En el panel de *Console Window* (CW) se muestran todos los mensajes y resultados de la síntesis, implementación, compilación, *warnings*, errores, y cualquier otro tipo de mensaje que permite visualizar los resultados de los diferentes procesos que están corriendo en ese momento.

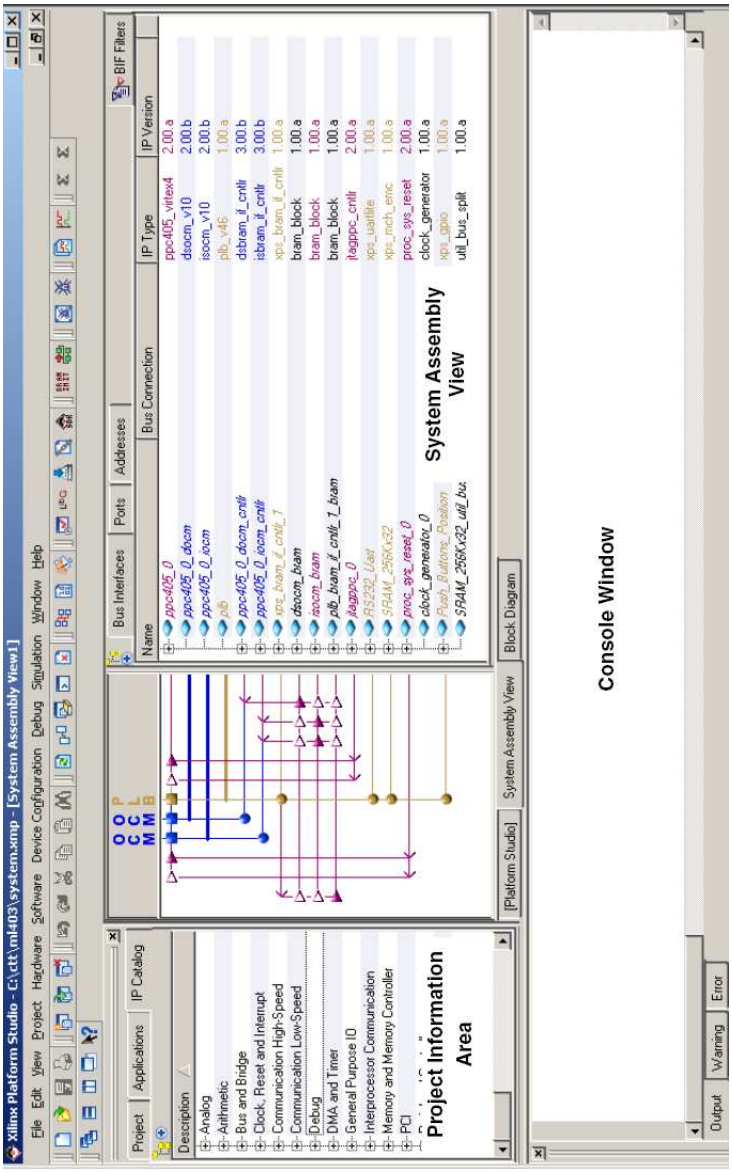


Figura A.1: Xilinx Platform Studio (XPS): Entorno de trabajo en EDK. Figura tomada de Xilinx-2 (Febrero 2008)

La forma de trabajo básicamente es ir construyendo desde cero la plataforma añadiendo cores desde la parte del PIA hacia el panel de SAV e ir realizando las interconexiones del sistema mediante el diagrama de buses que esta en la parte izquierda del SAV, después usando las pestañas de *Ports* y *Addresss* se conectan los puertos de cada *core* a los *buses* y se asignan sus direcciones de memoria respectivamente. A continuación se describe brevemente la realización de esta plataforma hardware usando la herramienta EDK.

El primer paso es colocar los *cores* necesarios de la plataforma, seleccionarlos desde la lista de IP *cores* y arrastrarlos al panel SAV, tal y como se observa en la figura A.2, en donde se ha añadido el procesador Microblaze, un bus OPB, un bus PLB y un *core* UART a manera de ejemplo. Una vez añadidos todos los *cores* que se necesitan se realiza una conexión "general" a los buses del sistema, teniendo como resultado un arquitectura global de la plataforma hardware.

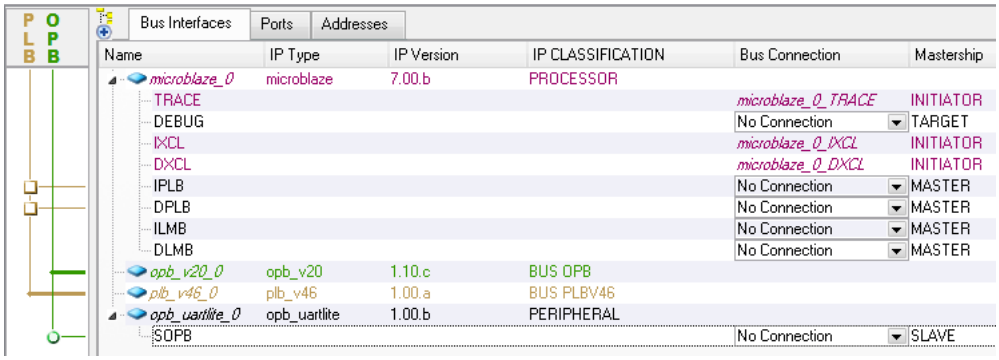


Figura A.2: Ejemplo de XPS de un sistema Microblaze con algunos cores añadidos a la plataforma hardware

Posteriormente en la pestaña de *Ports* se realizan las conexiones físicas de cada uno de las señales de E/S a bajo nivel de los *cores*, procesador y buses, esto se ilustra en la figura A.3, donde se observan algunos de las señales E/S del procesador Microblaze, en la columna *Net* es donde se indica a que señal interna o externa irán conectados cada uno de los puertos del procesador, este proceso se realiza para cada uno de los *cores* de la plataforma. Es responsabilidad del usuario realizar las conexiones de forma acorde y lógica a los requerimientos del sistema.

El siguiente paso es asignar los espacios de memoria para cada uno de los periféricos del sistema, ya que el procesador se comunica con sus periféricos por medio de direcciones de memoria, así que cada periférico deberá tener una única dirección de memoria para poder intercambiar datos con el procesador y los buses. La asignación de direcciones de memoria para cada periférico se observa en la figura A.4. También en este caso se puede decidir que tamaño de espacio de memoria se le asigna a cada periférico dependiendo de la cantidad de información que éstos vayan a manejar, un valor típico son 64 KB.

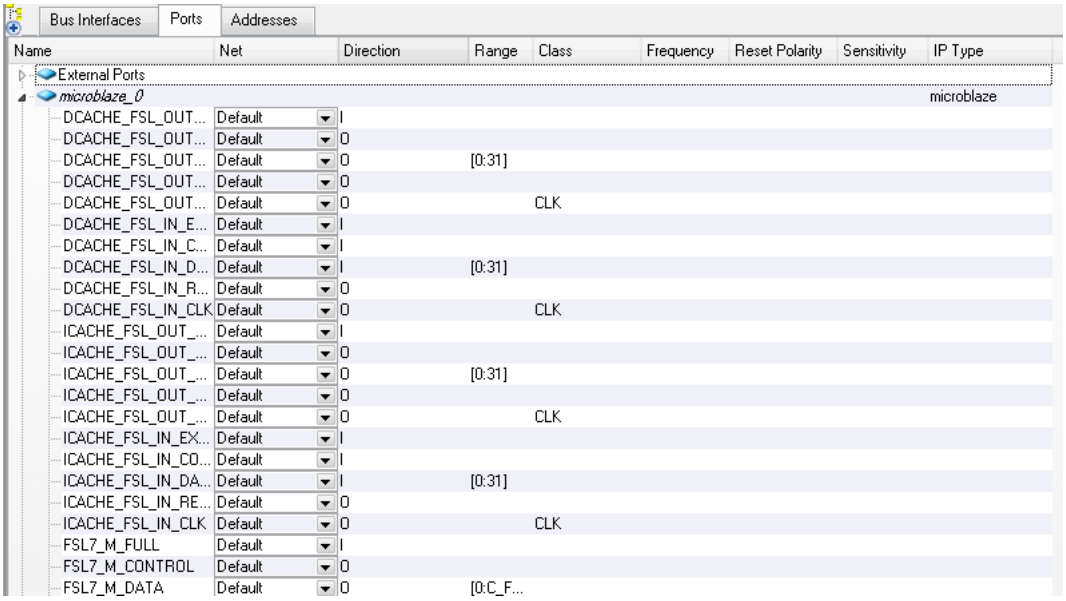
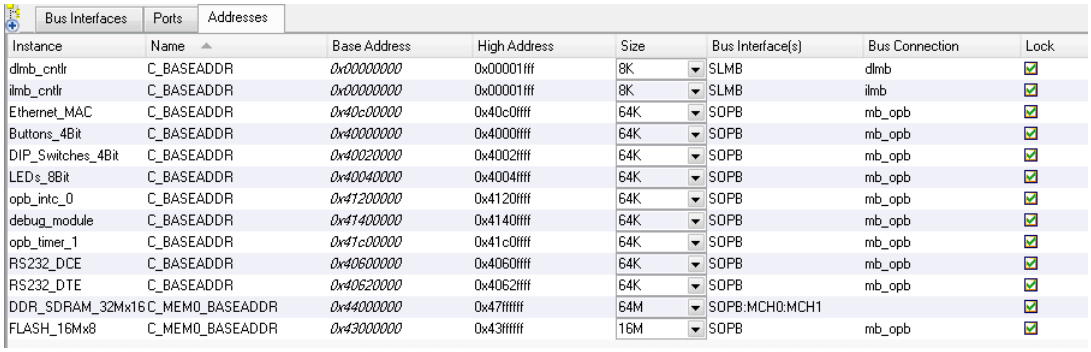


Figura A.3: XPS: Conexión de las señales E/S del procesador Microblaze



Bus Interfaces | Ports | Addresses

Instance	Name	Base Address	High Address	Size	Bus Interface(s)	Bus Connection	Lock
dlmb_cntlr	C_BASEADDR	0x00000000	0x00001fff	8K	SLMB	dlmb	☑
ilmb_cntlr	C_BASEADDR	0x00000000	0x00001fff	8K	SLMB	ilmb	☑
Ethernet_MAC	C_BASEADDR	0x40c00000	0x40c0ffff	64K	SOPB	mb_opb	☑
Buttons_4Bit	C_BASEADDR	0x40000000	0x4000ffff	64K	SOPB	mb_opb	☑
DIP_Switches_4Bit	C_BASEADDR	0x40020000	0x4002ffff	64K	SOPB	mb_opb	☑
LEDs_8Bit	C_BASEADDR	0x40040000	0x4004ffff	64K	SOPB	mb_opb	☑
opb_intc_0	C_BASEADDR	0x41200000	0x4120ffff	64K	SOPB	mb_opb	☑
debug_module	C_BASEADDR	0x41400000	0x4140ffff	64K	SOPB	mb_opb	☑
opb_timer_1	C_BASEADDR	0x41c00000	0x41c0ffff	64K	SOPB	mb_opb	☑
RS232_DCE	C_BASEADDR	0x40600000	0x4060ffff	64K	SOPB	mb_opb	☑
RS232_DTE	C_BASEADDR	0x40620000	0x4062ffff	64K	SOPB	mb_opb	☑
DDR_SDRAM_32Mx16	C_MEM0_BASEADDR	0x44000000	0x47ffffff	64M	SOPB:MCH0:MCH1		☑
FLASH_16Mx8	C_MEM0_BASEADDR	0x43000000	0x43ffffff	16M	SOPB	mb_opb	☑

Figura A.4: XPS: asignación de espacio de memoria a los *cores* del sistema

A.2. Desarrollo del software del sistema

Una vez que se ha concluido la realización de la parte hardware del sistema, ahora se tiene que realizar la parte software del mismo, es decir, los programas de usuario que estará corriendo el procesador. Antes de crear cualquier aplicación de usuario es necesario generar toda una infraestructura software que se comunique a bajo nivel con la plataforma hardware, este puente entre en hardware y software se llama *Board Support Package* (BSP).

El BSP es una colección de archivos que definen, para cada procesador, los elementos hardware del sistema. El BSP también contiene información de los elementos software, tal como los controladores para cada periférico, bibliotecas, rutinas de interrupción, dispositivos estándar de E/S, y otras características relacionadas; el BSP es el que permite cargar un SO sobre la plataforma hardware.

Existe una utilidad llamada LibGen la cual permite generar el BSP, el LibGen toma el archivo MSS para obtener toda la información necesaria.

Todas las características seleccionadas para el software del sistema se configuran en la opción de *Software Platform Settings*, figura A.5 y se almacenan en el archivo MSS, que junto con las aplicaciones de usuario representan la parte software completa de todo el sistema. En esta interfaz se define la frecuencia de reloj del sistema, el tipo de SO, la versión de los controladores para cada periférico, entre otras características. Dentro de los posibles SO que se pueden utilizar se tiene el *Stand-Alone*, que en realidad no es un SO en si, más bien son solamente las funciones que Xilinx pone a disposición para poder usar sus periféricos a bajo nivel. Existe una gran variedad de SO disponibles por otros fabricantes (*third party*), algunos son: Petalinux, BlueCat, uC/OS-II, Nucleus, MontaVista Linux, LynuxOS, etc.

Toda la información de controladores, bibliotecas y demás archivos software permiten compilar las aplicaciones de usuario en un archivo llamado ELF (*Excecutable and Linkable Format*). El archivo ELF es un archivo binario que corre en el procesador del sistema, en la figura A.6 se observa la generación de este archivo.

El resultado de sintetizar la parte hardware del diseño se encuentra en un archivo *bitstream* llamado 'system.bit', mientras que el resultado de compilar la parte software del diseño se tiene en el archivo ELF con extensión *.elf. Una vez que se tiene la información en archivos tanto de la parte hardware como la software, el siguiente paso es fusionar estos dos archivos en uno solo para programar el FPGA, este archivo bitstream se llama 'download.bit'. Mientras que la utilidad que permite esta fusión de archivos se llama *Data2Mem*, con esto se obtiene el archivo final que contiene la información de ambas partes (HW/SW).

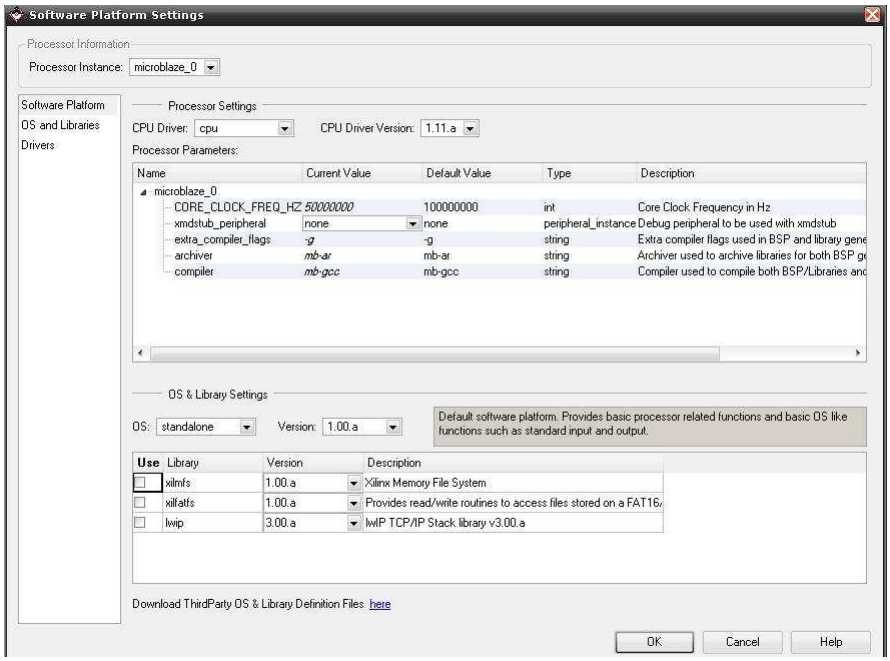


Figura A.5: Cuadro de diálogo para configurar la parte Software del sistema. En este cuadro pueden configurarse el tipo de SO que se usa, así como la versión de los controladores para los distintos periféricos del sistema

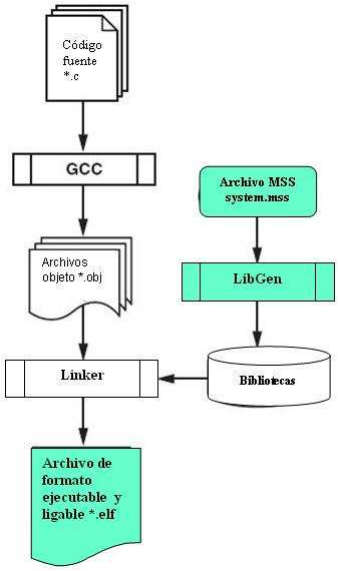


Figura A.6: Generación del archivo ejecutable ELF. La herramienta *Lib generator* se encarga de generar las bilibotecas necesarias apartir del cuadro de configuración del sistema. Figura tomada de Xilinx-2 (Febrero 2008)

A.3. Periféricos del sistema

En esta sección se presentan primeramente los periféricos (sensores/actuadores) que se añadieron al sistema Microblaze. Dado que los sensores requieren una interfaz hardware y con-

troladores software, se describe a continuación como se procedió para incorporar los sensores/actuadores particulares del robot móvil de prueba. Previamente se da una breve descripción de las características de estos sensores/actuadores.

A.3.1. Sonares

Para este trabajo se emplearon tres tipos distintos de sonares: SRF05, PING y SRF02 (Acroname, Diciembre 2008). A continuación se definen brevemente cada uno de ellos:

SRF05. Sonar con un rango de detección de 2 cm hasta 4 mts. Las señales que maneja son dos: una entrada de eco, a la cual se tiene que mandar un pulso (10 useg aprox.) para que el sonar emita la señal de ultrasonido y una salida de eco, la cual devuelve un ancho de pulso proporcional a la distancia. .

PING. Sonar con un rango de detección de 2 cm hasta 3 mts. Las señales que maneja es solo una: primero se manda un pulso (5 useg aprox.) se espera un tiempo en lo que llega la respuesta (750 useg) y finalmente se lee el resultado en forma de ancho de pulso, igual que el SRF05.

SRF02. Sonar con un rango de detección de 15 cm hasta 3 mts. Maneja un único transductor tanto para la emisión como para la recepción, a diferencia de los dos anteriores que tienen transductores separados. El manejo de señales es por medio del bus I2C, así que para obtener el resultado, el cual ya esta dado en cm, se debe hacer leyendo un registro del dispositivo, esto respetando el protocolo I2C. La implementación de este tipo de sonar se presenta en la sección A.3.2.

La finalidad ahora es crear un periférico hecho por el usuario (*custom* IP), el cual realice la "traducción" de ancho de pulso a un número binario que represente la distancia en cm. Para lograr esto EDK utiliza la librería de Interfaz para Propiedad Intelectual (IPIF). IPIF sirve para implementar distintas funcionalidades en hardware para diferentes tipos de procesadores, funcionalidades como el paso de datos, escritura/lectura de FIFOS, control de interrupciones, decodificación de direcciones, control de señales maestro/esclavo, señales para comunicación con el Bus OPB/PLB, etc. Esta interfaz ya se encuentra probada, optimizada y es altamente parametrizable. IPIF también ofrece un protocolo de bus simplificado llamado IP *Interconnect* (IPIC), que sirve para comunicar la lógica de usuario con el módulo IPIF, este protocolo es mucho más sencillo que manejar directamente la interfaz con los buses OPB o PLB.

En la figura A.7 se observa el diagrama a bloques del periférico completo para trabajar con el sonar SRF05. La funcionalidad del módulo SRF05, que es la lógica de usuario, es mandar un pulso a la entrada de *reset*, para que automáticamente se mande un pulso de salida para activar la señal *burst* del sonar, una vez que la señal emitida por el sonar rebota y el eco de entrada (ancho de pulso) llega, ahora comienza el proceso de conversión, dando finalmente como resultado la salida 'salcm' (resultado de la distancia en cm). También se observa el periférico completo conectado al bus PLB del procesador, aquí se observa el módulo hecho por el usuario junto con la interfaz IPIF y las señales externas del periférico.

La única diferencia para el PING *ultrasonic* es que se implementa una lógica de salida mediante un *buffer* para que por el mismo pin se implementen las señales de entrada y salida de eco.

Por lo que respecta al sonar SRF02, ya que trabaja con el bus I2C, este tipo de sonar se presenta en la siguiente sección.

Como puede observarse en los diagramas hardware para los sonares, existen unos nombres llamados 'slv_reg', que en realidad son registros que pueden accederse por medio de funciones software, estos registros son los que van ayudar a activar y leer el resultado de los sonares a través de aplicaciones de usuario. De esta manera, la única tarea del usuario es activar la señal

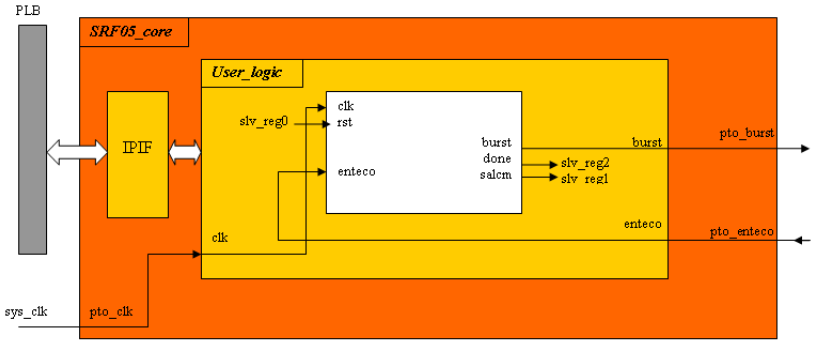


Figura A.7: Diagrama para el módulo hardware del sonar SRF05 y el módulo de interconexión con el bus PLB

de *reset* escribiendo al registro_0 un 1, esperar a que el resultado de los sonares este listo, leyendo la señal *'done'* (registro slv_reg2), y una vez que el resultado está listo, simplemente leer el registro_1 el cual siempre contiene el resultado en cm. Las funciones disponibles para este tipo de periféricos se encuentran en la tabla B.4.

A.3.2. Bus I2C

Para este trabajo se emplearon dos dispositivos que trabajan mediante bus I2C: la brújula CMPS03 y el sonar SRF02 (ver figura A.8). A continuación se describe brevemente el funcionamiento de cada uno de estos sensores:

SRF02. Sonar cuyo rango de detección esta entre 15 cm y 4 mts. Puede dar una salida en uSeg, cm o pulgadas. Su dirección de fábrica es 0xE0. Sus señales de control son SCL y SDA. Posee 6 registros, dentro de los que destacan el registro 0x02 y 0x03 los cuales son el byte más significativo y menos significativo respectivamente para la lectura de la distancia.
La secuencia de funcionamiento que se debe implementar a nivel software es la siguiente:
- Escribir la dirección del dispositivo 0x0E para establecer comunicación
- Escribir al registro de comandos (0x00) el modo de medición que se desea: uS (0x50), cm (0x51), in (0x52).
- Esperar un retardo de 70 uSeg aproximadamente para esperar a que el eco regrese al sensor.
- Leer los registros 0x02 y 0x03 para el resultado de la medición, un valor de 0 devuelto indica que ningún objeto fue detectado.

CMPS03. Brújula usado especialmente para tareas de navegación en Robótica Móvil, se usa para detectar la dirección hacia la cual el robot esta apuntando. Sus señales de control son SCL y SDA. Aunque también puede producir una señal PWM, pero no es usada para este trabajo. Su dirección de fábrica es 0xC0. Posee 16 registros para su funcionamiento dentro de los que destacan los registros 0x01, el cual devuelve la dirección en un rango de [0 255], los registros 0x02 y 0x03 devuelven la dirección en un rango de [0 3599], representando [0°359.9°] y el registro 0x0F para calibrar la brújula
La secuencia que se debe seguir a nivel software para calibrar la brújula es:
- Colocar la brújula apuntando hacia el Norte y escribir el valor 255 al registro 0x0F
- Colocar la brújula apuntando hacia el Este y escribir el valor 255 al registro 0x0F
- Colocar la brújula apuntando hacia el Sur y escribir el valor 255 al registro 0x0F
- Colocar la brújula apuntando hacia el Oeste y escribir el valor 255 al registro 0x0F

La secuencia de funcionamiento que se debe implementar a nivel software para la lectura de la dirección a la que apunta la brújula es la siguiente:
- Escribir la dirección del dispositivo 0xC0 para establecer comunicación
- Leer los registros 0x01 o bien 0x02 y 0x03 para obtener el resultado

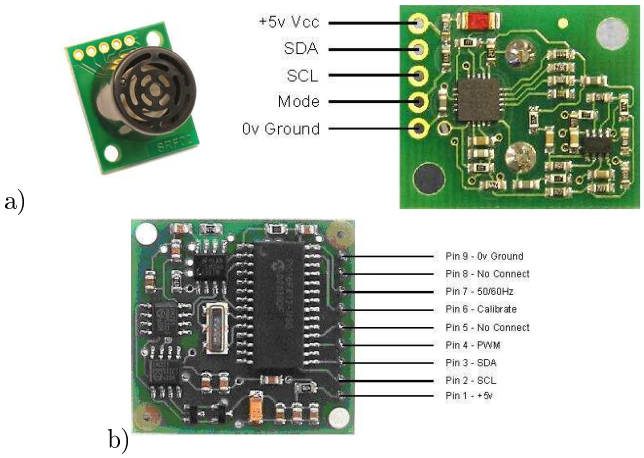


Figura A.8: a) Sonar SRF02 y b)Brújula CMPS03. Se muestran las señales de E/S para cada dispositivo. Figura tomada de Acroname (Diciembre 2008)

A.3.3. Sensores infrarrojos

Los sensores infrarrojos usados para este trabajo son: el sensor de proximidad GP2D15 y sensor seguidor de línea de LynxMotion (ver A.9). A continuación se describe brevemente su funcionamiento:

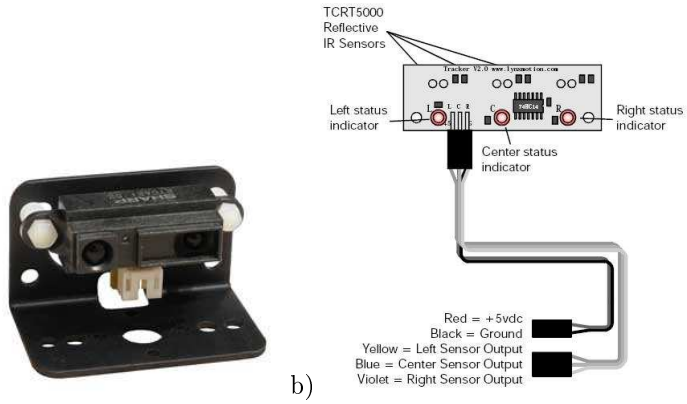
GP2D15. Sensor de proximidad con salida digital, coloca un 1 a la salida si el objeto se encuentra en el rango de 10 cm a 80 cm y 0 en caso contrario. Solo tiene esta salida de control la cual indica si el objeto esta presente o no.

TRACKING LINE. Sensor con un arreglo de 3 sensores IR reflectivos colocados uniformemente en una línea horizontal, cada uno de los sensores reflectivos ofrece una salida binaria de 0 si no hay reflejo de la señal y de 1 si hay reflejo.

Para la implementación de este tipo de *cores*, simplemente se añadieron *cores* tipo GPIO, los cuales ya vienen como parte de la herramienta EDK. Es un *core* sencillo de implementar, solamente es necesario unos cuantos registros (*Flip-flops*), *buffer* tri-estado y algunos multiplexores para el control de flujo de las señales. Este tipo de *core* esta diseñado para trabajar con señales de entrada y salida, además de que el ancho de palabra de datos es parametrizable por el usuario.

A.3.4. Brazo manipulador AX-12

Un dispositivo bastante útil en Robótica Móvil para aplicaciones donde se requieren tomar o transportar objetos es el Brazo Manipulador. En este trabajo se utiliza el Brazo AX-12 de la compañía CrustCrawler (CrustCrawler, Agosto 2008). Este brazo tiene la característica de trabajar con motores servo llamados *Dynamixel Actuators*, de la misma compañía. Las principales características de estos actuadores son:

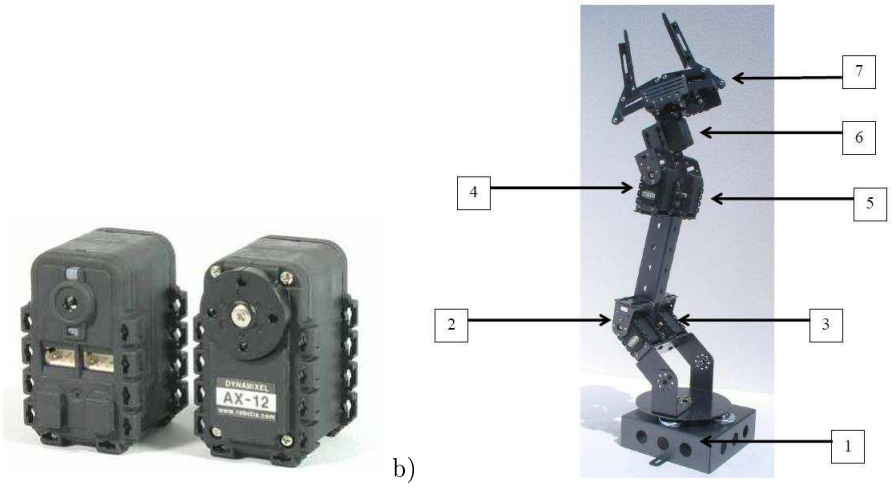


a) b)

Figura A.9: Sensores infrarojos usados para este trabjao. a) Sensor de Proximidad y b) Sensor Seguidor de Línea. Figuras tomadas de Lynxmotion (Diciembre 2008)

- Tiene una resolución de movimiento 0.35°
- Tiene una movimiento de 300°, o movimiento libre
- Comunicación serial desde 7343 bps hasta 1 Mbps.
- Forma de interconexión entre Dynamixels tipo *daisy chain*.
- Información de estado de posición, temperatura, voltaje, etc.
- Comunicación serial tipo *half-duplex*
- 1024 pasos de movimiento del servo, rango [0 1023] representando [0 300°] respectivamente

Como se puede observar en la figura A.10, el brazo consta de 7 Dynamixels conectados en *daisy chain*, y numerados de abajo hacia arriba. Cada uno de los Dynamixels se comunica por medio de un protocolo serial tipo *half duplex* a una velocidad típica de 1 Mbps.



a) b)

Figura A.10: a)Actuadores Dynamixels y b)Brazo Manipulador AX-12. El brazo manipulador esta conformado por siete servos dynamixels. Figuras tomadas de CrustCrawler (Agosto 2008)

Para la implementación del *core* que pueda controlar el AX-12, se recurre al *core* UART16550, el cual es un *core* que implementa la comunicación serial tipo *full-duplex* con la característica principal de que el *Baud Rate* es seleccionado por software, escribiendo a cierto registro de configuración. Ya que el UART usado es de tipo *full-duplex* es necesario implementar una lógica de *buffers* tri-estado para convertir la comunicación a *half-duplex*. En la figura A.11 se observa el diagrama de conexión entre el FPGA y el Brazo AX-12.

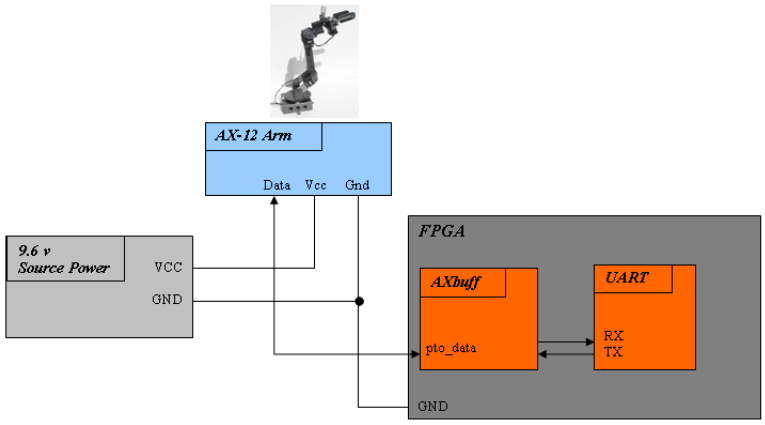
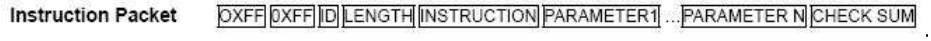


Figura A.11: Diagrama de conexión para el brazo robot AX-12. El kit del brazo AX-12 viene con una batería con la cual se puede alimentar al brazo robot

Algo importante a resaltar es que se nombraron los dynamixels de la siguiente manera: #1 se le llama "*Base*", al #2 y #3 se le llama "*Join1*", al #4 y #5 se le llama "*Join2*", al #6 se le llama "*Wrist*" y al #7 se le llama "*Gripper*".

Para controlar el movimiento de cada Dynamixel se escriben valores en sus registros, estos registros permiten verificar el estado de cada dynamixel así como controlar su movimiento, dentro de los registros importantes se tienen: el 0x03 y 0x04 que controlan el identificador ID y el *Baud Rate* respectivamente, se tiene el registro 0x0E que controla el torque, el 0x1E y 0x20 que controlan la posición y la velocidad de movimiento respectivamente. En la tabla A.1 se observan solo algunos de los registros más importantes junto con los valores permitidos de escritura. En total cada dynamixel consta de 50 registros para el control de sus características.

A continuación se muestra la forma del paquete de instrucción que se tiene que mandar a los Dynamixels:



ID es el identificador de cada dynamixel, los cuales van del 1 al 7. LENGTH esta dato por el número de parámetros + 2. INSTRUCTION es el tipo de instrucción que se manda al dynamixel, donde las instrucciones pueden ser: PING, READ_DATA, WRITE_DATA, REG_WRITE, ACTION, RESET y SYNC_WRITE. Después viene la lista de parámetros dependiendo de lo que se quiera escribir en los registros y del tipo de instrucción. Finalmente se tiene el CHECKSUM el cual esta dado por ~ (ID + LENGTH + INST + PARAM_1 + ... + PARAM_n), donde '~' significa la negación lógica de la suma. Todos los campos del paquete de instrucción son de 1 byte.

A.3.5. Co-procesador para imágenes

La herramienta utilizada para implementar este *core* se llama Xilinx System Generator (XSG), la cual es otra de las herramientas de Xilinx, especialmente diseñada para trabajar con

Write Address	Writing Item	Length (bytes)	Min	Max
3(0X03)	ID	1	0	253(0xfd)
4(0X04)	Baud Rate	1	0	254(0xfe)
5(0X05)	Return Delay Time	1	0	254(0xfe)
6(0X06)	CW Angle Limit	2	0	1023(0x3ff)
8(0X08)	CCW Angle Limit	2	0	1023(0x3ff)
11(0X0B)	the Highest Limit Temperature	1	0	150(0x96)
12(0X0C)	the Lowest Limit Voltage	1	50(0x32)	250(0xfa)
13(0X0D)	the Highest Limit Voltage	1	50(0x32)	250(0xfa)
14(0X0E)	Max Torque	2	0	1023(0x3ff)
16(0X10)	Status Return Level	1	0	2
17(0X11)	Alarm LED	1	0	127(0x7f)
18(0X12)	Alarm Shutdown	1	0	127(0x7f)
19(0X13)	(Reserved)	1	0	1
24(0X18)	Torque Enable	1	0	1
25(0X19)	LED	1	0	1
26(0X1A)	CW Compliance Margin	1	0	254(0xfe)
27(0X1B)	CCW Compliance Margin	1	0	254(0xfe)
28(0X1C)	CW Compliance Slope	1	1	254(0xfe)
29(0X1D)	CCW Compliance Slope	1	1	254(0xfe)
30(0X1E)	Goal Position	2	0	1023(0x3ff)
32(0X20)	Moving Speed	2	0	1023(0x3ff)
34(0X22)	Torque Limit	2	0	1023(0x3ff)
44(0X2C)	Registered Instruction	1	0	1
47(0X2F)	Lock	1	1	1
48(0X30)	Punch	2	0	1023(0x3ff)

Tabla A.1: Tabla de control con algunos de los registros para los Dynamixels

aplicaciones de procesamiento digital de señales (DSP). XSG es un ambiente de diseño integrado (IDE) a nivel sistema para FPGAs, que utiliza a Simulink de MATLAB©como entorno de desarrollo y se hace presente en forma de *blockset*. La idea es utilizar a XSG para diseñar el *core* de imágenes y después importarlo a EDK como una caja negra para poder utilizarlo como periférico. El tipo de interfaz usada para conectar este periférico al procesador Microblaze es el FSL, tal y como se muestra en la figura A.12.

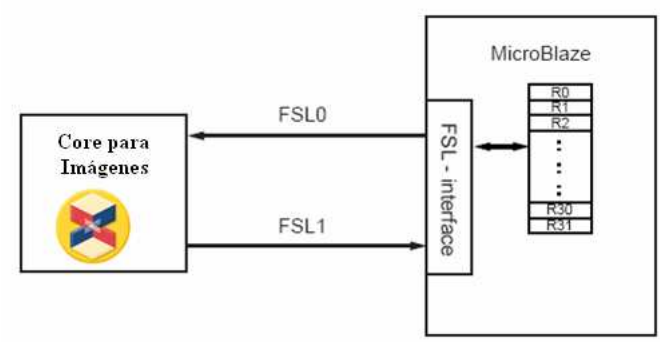


Figura A.12: Co-procesador para imágenes conectado a la interfaz FSL (*Fast Simple Link*) del procesador Microblaze. Figura tomada de Xapp-Xilinx (2004)

Este *core* tiene la característica de ser parametrizable por el usuario, dentro de estos parámetros se tiene principalmente el tamaño de la imagen, el tamaño de la máscara, la representación de los coeficientes de la máscara, entre otros. Esto es de bastante utilidad ya que el usuario puede aplicar cualquier tipo de máscara de convolución a la imagen.

Para que la arquitectura sea parametrizable se utilizaron dos conceptos importantes en

MATLAB©:

Generación Dinámica de bloques. La forma común de trabajar con Simulink es la esquemática, arrastrando y conectando bloques. Pero existe otra manera poco usual de construir estos mismos modelos, y es mediante programación dinámica, esto se logra por medio de la API (*Application Programming Interface*) de MATLAB©, la cual da soporte a instanciación de bloques y señales, además de configuración, realización y eliminación de bloques, entre otros métodos de construcción. Usando esta API, es posible escribir *scripts* de MATLAB©que generen modelos completos de Simulink y por tanto, también modelos de XSG. Así, este tipo de técnica permite modificar la estructura de un subsistema en respuesta a los cambios realizados a ciertos parámetros.

Enmascaramiento de subsistemas. Un subsistema en Simulink es el agrupamiento de varios bloques con sus correspondientes conexiones en uno solo, formando así una "caja negra" con sus correspondientes entradas y salidas. Esto da una gran versatilidad al diseño y reduce considerablemente el espacio de trabajo en Simulink, al mismo tiempo que da una mejor organización jerárquica del diseño. Por otro lado enmascarar un subsistema se refiere a asignarle a esta "caja negra" un cuadro de diálogo personalizado que permita configurar diferentes parámetros de los bloques que están dentro de ese subsistema. Esta operación se realiza por medio de la herramienta *Mask Editor* de Simulink.

De esta manera se puede crear el *core* según los requerimientos del usuario antes de ser sintetizado e importarlo a la herramienta EDK. La figura A.13 muestra el *core* completo como una caja negra así como el cuadro de diálogo que lo configura.

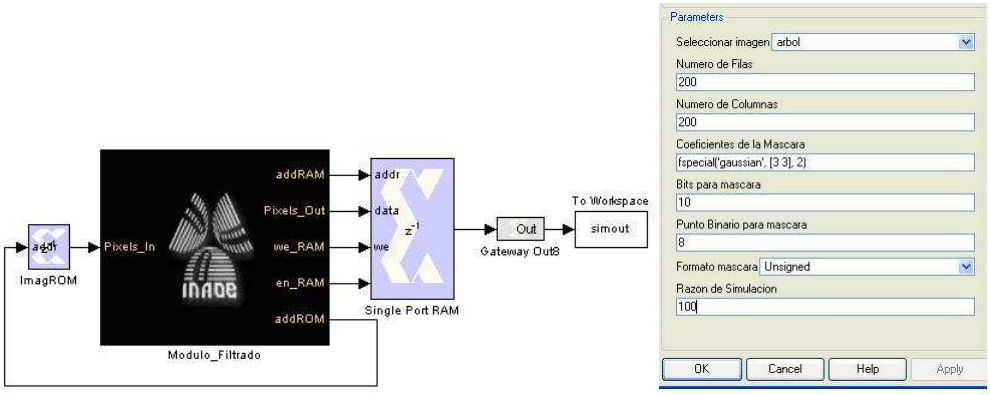
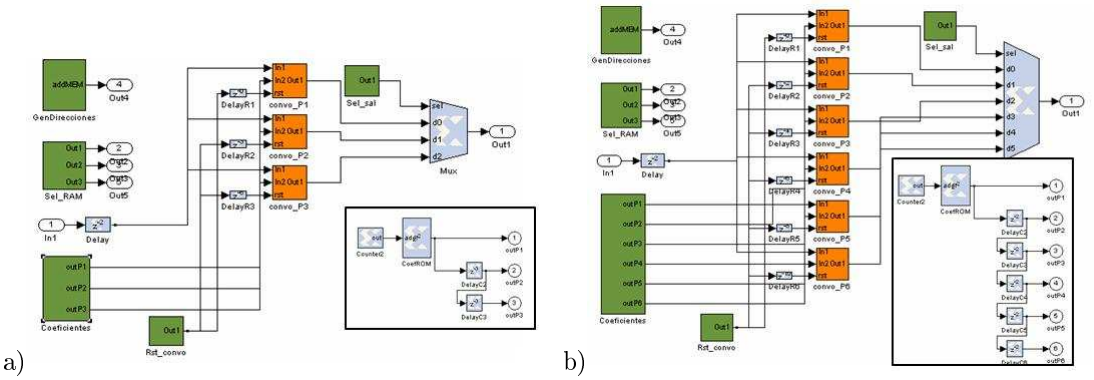


Figura A.13: Subsistema del core para imágenes y su cuadro de diálogo, por medio del cual el usuario puede configurar distintos parámetros de la arquitectura, como el tamaño de la máscara sus coeficientes y la representación de los coeficientes

En la figura A.14 se observa la arquitectura dentro del subsistema y de como ésta va cambiando conforme el usuario selecciona parámetros distintos. En este ejemplo se observa la arquitectura generada automáticamente para a) una máscara de convolución de 3x3 y b) una máscara de 5x5.

Una vez que se tiene el core configurado a las necesidades del usuario, el siguiente paso es importarlo como periférico al sistema empotrado Microblaze. Esto se realiza por medio de una utilidad de la herramienta EDK especialmente diseñada para importar co-procesadores, este utilidad funciona por medio de un *wizard* que va llevando paso a paso la configuración del co-procesador. De esta manera una vez hechas todas las conexiones necesarias y tener el co-procesador listo en el sistema Microblaze, solo resta colocar los datos de entrada y leer los resultados mediante dos instrucciones simples que se usan desde la aplicación software.



a) b)

Figura A.14: Efecto de la programación dinámica de bloques en el módulo co-procesador para imágenes

Apéndice B

Funciones software para el robot de prueba

Las bibliotecas de funciones presentadas a continuación forman parte de la arquitectura software vista en la sección 4.3. Dichas bibliotecas fueron usadas para validar el robot de prueba presentado para este trabajo. El usuario también puede usar estas bibliotecas para realizar sus propias aplicaciones robóticas.

B.1. Biblioteca: Robot móvil Create

A continuación se enlistan las funciones en las diferentes capas de abstracción para el robot móvil Create.

Función Capa Drivers	**Descripción**
void XUartLite_SendByte (u32 BaseAddress, u8 Data)	Transmite un byte por el módulo UART
u8 XUartLite_RecvByte (u32 BaseAddress)	Recibe un byte por el módulo UART
int XUartLite_Initialize (XUartLite *InstancePtr, u16 DeviceId)	Inicializa el dispositivo UART a sus valores por defecto
void XUartLite_ResetFifos (XUartLite *InstancePtr)	Limpia las FIFOs tanto del transmisor como del receptor
unsigned int XUartLite_Send (XUartLite *InstancePtr, u8 *DataBufferPtr, unsigned int NumBytes)	Envia cierto número de bytes
unsigned int XUartLite_Recv (XUartLite *InstancePtr, u8 *DataBufferPtr, unsigned int NumBytes)	Recibe cierto número de bytes
int XUartLite_IsSending (XUartLite *InstancePtr)	Indica si el buffer del transmisor esta o no vacío
XUartLite_EnableInterrupt (XUartLite *InstancePtr)	Habilita la interrupción del módulo

Tabla B.1: Funciones de la capa controladores para el robot móvil Create, básicamente son funciones para el control del módulo serial UART.

Función capa Básica	**Descripción**
void send_com(Xuint8 vec[],Xuint8 tam)	Manda comando al Robot Create
void send_com2(Xint16 *vec)	Manda comando al Robot mediante apuntador
void flush_uart()	Limpia el Transmisor del módulo UART
void wait_dist(Xint16 dist)	Espera a que el robot recorra una distancia
void wait_angle(Xint16 ang)	Espera a que el robot gire cierto ángulo
void wait_time(Xuint8 time)	Espera a que pase cierto tiempo
void wait_event(Xuint8 event)	Espera a que pase algún evento
void walk(Xint16 vel)	Aplica la misma velocidad a ambas ruedas
void walk2(Xint16 vel_i, Xint16 vel_d)	Aplica distinta velocidad a cada rueda
void stop_robot(void)	Detiene el robot
void turn(Xint16 vel)	Gira el robot a cierta velocidad sobre su propio eje
void turn_angle(Xint16 angle, Xuint8 vel)	Gira el robot determinado ángulo y a cierta velocidad
void led_manage(Xuint8 op,Xuint8 color,Xuint8 intensity)	Manipula los LEDs de Play y Advanced, así como da intensidad al LED de Power
void play_song(Xuint8 vec[],Xuint8 tam)	Emite serie de sonidos
void send_IR(Xuint8 val)	Recibe información del sensor Infrarrojo
void send_Dout(Xuint8 val)	Envia el valor por el puerto digital del robot
void script(Xuint8 len)	Envia un pequeño script al robot
void play_script()	Ejecuta el script
Xuint8 is_OneByte(Xuint8 pack)	Indica si el paquete a recibir es de uno o dos bytes
void send_sensor(Xuint8 pack)	Manda petición del paquete hacia el robot
Xint16 receive_sensor(Xuint8 num)	Recibe la petición del paquete
Xint16 request_sensor(Xuint8 pack)	Recibe la información del sensor solicitado
Xint16 receive_stream(Xuint8 num)	Recibe información de cierto paquete de forma continua

Tabla B.2: Funciones de la capa básica para el robot móvil Create

Función	Descripción
void Advance(Xint16 vel)	Avanza el robot a cierta velocidad
void Bumper_stop()	Si se activa algún bumper, detiene el robot
void Bumper_back (Xuint8 dist)	Si se activa algún bumper, retrocede cierta distancia
void Bumper_back_turn (Xuint8 dist, Xint16 angle)	Si se activa algún bumper, retrocede cierta distancia y gira determinados grados
void Bumper_avoid (Xuint8 dist1, Xint16 angle1, Xuint8 dist2, Xint16 angle2)	Si se activa algún bumper, trata de rodear el obstáculo
void Bumper_avoid90 (Xuint8 dist)	Si se activa algún bumper, evade el obstáculo con ángulos de 90 grados
void Bumper_action(Xuint8 side, Xuint8 action)	Dependiendo que lado se active el bumper realiza alguna acción predeterminada
void IR_action(Xuint8 data, Xuint8 action)	Dependiendo el dato que reciba del sensor Infrarrojo, realiza determinada acción
void Detect_OverCurrent()	Detecta si existe un sobreflujo de corriente en las ruedas
void Detect_OverTemp()	Detecta si existe un sobrecalentamiento en la batería del robot Create

Tabla B.3: Funciones para la capa Intermedia del robot móvil Create

B.2. Biblioteca: Sonares

A continuación se enlistan las funciones software para los sonares, en todos sus niveles de abstracción.

Función Capa Drivers	**Descripción**
SONARES_mWriteReg (BaseAddress, RegOffset, Data)	Escribe a algún registro del módulo dependiendo el off-set
SONARES_mReadReg (BaseAddress, RegOffset)	Lee a algún registro del módulo dependiendo el off-set
SONARES_mWriteSlaveReg-'n' (BaseAddress, Value)	Escribe al registro 'n'
SONARES_mReadSlaveReg-'n' (BaseAddress)	Lee al registro 'n'
SONARES_mReset (BaseAddress)	Limpia los registros e inicializa el módulo
SONARES_SelfTest (void * baseaddr_p)	Test de prueba para el módulo de sonares
Función Capa Básica	**Descripción**
int ReadSensor16_SRF02 (Xuint32 IicBaseAddress, Xuint8 SensorAddress, Xuint8 RegAdd, Xuint8 numBytes)	Lee el sonar SRF02 el cual funciona bajo el bus I2C, esta función requiere de algunos parámetros necesarios para el bus I2C
int Read_SRF02()	Devuelve el dato del sonar SRF02 en cm
int Read_SRF05()	Devuelve el dato del sonar SRF05 en cm
int Read_PING()	Devuelve el dato del sonar PING en cm
Función Capa Intermedia	**Descripción**
int Burst_Sonar()	Activa la lectura de todos los sonares, esta función hace que los sonares envíen el pulso de ultrasonido
int Read_Sonar(Xuint8 numsonar)	Devuelve el valor en cm de cualquiera de los sonares, dependiendo lo que tenga 'numsonar'

Tabla B.4: Funciones para los sonares, incluyendo la capa controladores, básica e intermedia

B.3. Biblioteca: Brújula

A continuación se enlistan las funciones software para la brújula la cual funciona mediante un módulo I2C, las funciones presentadas abarcan distintos niveles de abstracción.

Función Capa Controladores	**Descripción**
int XIic_Initialize (XIic *InstancePtr, u16 DeviceId)	Inicializa con los valores por default los parámetros del bus I2C
void XIic_Reset (XIic *InstancePtr)	Limpia los registros del bus
void XIic_SetAddress (XIic *InstancePtr, int AddressType, int Address)	Coloca una dirección a un dispositivo esclavo
U16 XIic_GetAddress (XIic *InstancePtr, int AddressType)	Obtiene la dirección de un dispositivo que se encuentre conectado al bus
u8 XIic_Send (u32 BaseAddress, u8 Address, u8 *BufferPtr, unsigned ByteCount, u8 Option)	Manda datos por el bus I2C al dispositivo con cierta dirección
u8 XIic_Recv (u32 BaseAddress, u8 Address, u8 *BufferPtr, unsigned ByteCount, u8 Option)	Recibe datos por el bus I2C desde un dispositivo con cierta dirección
Función Capa Básica	**Descripción**
Xuint8 ReadSensor8_COMPASS (Xuint32 IicBaseAddress, Xuint8 SensorAddress, Xuint8 RegAdd, Xuint8 numBytes)	Petición de datos a la brújula por el bus I2C en formato de 1 byte
int ReadSensor16_COMPASS (Xuint32 IicBaseAddress, Xuint8 SensorAddress, Xuint8 RegAdd, Xuint8 numBytes)	Petición de datos al brújula por el bus I2C en formato de 2 bytes
Xuint8 Read8_Compass()	Obtiene la lectura de la brújula en un rango de [0 255]
Xuint8 Read16_Compass()	Obtiene la lectura de la brújula en un rango de [0 3599]

Tabla B.5: Funciones para el módulo I2C y el sensor brújula CMPS03, incluyendo las capas de controladores, básica e intermedia

B.4. Biblioteca: Sensores infrarrojos

A continuación se enlistan las funciones software para el control de los sensores infrarrojos, las funciones abarcan distintos niveles de abstracción.

Función Controladores	**Descripción**
void XGpio_mWriteReg (BaseAddress, RegOffset, Data)	Escribe a un registro en específico del canal 1 dependiendo del offset
u32 XGpio_mReadReg (BaseAddress, RegOffset)	Lee a un registro en específico del canal 1 dependiendo del offset
void XGpio_mSetDataDirection (BaseAddress, Channel, DirectionMask)	Indica mediante una máscara binaria la dirección de los pines (entrada o salida)
u32 XGpio_mGetDataReg (BaseAddress, Channel)	Obtiene el valor de un registro en un canal determinado
void XGpio_mSetDataReg (BaseAddress, Channel, Data)	Coloca el valor de un registro en un canal determinado
int XGpio_Initialize (XGpio *InstancePtr, u16 DeviceId)	Inicializa el módulo de pines de propósito general
int XGpio_SelfTest (XGpio *InstancePtr)	Prueba para el funcionamiento del módulo
void XGpio_InterruptEnable (XGpio *InstancePtr, u32 Mask)	Habilita la interrupción del módulo
void XGpio_InterruptDisable (XGpio *InstancePtr, u32 Mask);	Deshabilita la interrupción del módulo
Función Capa Básica	**Descripción**
u32 XGpio_DiscreteRead (XGpio *InstancePtr, unsigned Channel)	Lee los valores que llegan al puerto E/S
void XGpio_DiscreteWrite (XGpio *InstancePtr, unsigned Channel, u32 Mask);	Escribe valores al puerto E/S
int Read_LineTrack()	Obtiene el valor del sensor de seguidor de línea
int Read_Proximity()	Obtiene el valor del sensor de proximidad

Tabla B.6: Funciones para los sensores infrarrojos, incluyendo las capas de controladores, básica e intermedia

B.5. Biblioteca: Brazo robot

A continuación se enlistan las funciones software para el control del brazo robot, las funciones abarcan distintos niveles de abstracción.

Función Capa Controladores	**Descripción**
void XUartNs550_SendByte (u32 BaseAddress, u8 Data)	Transmite un byte por el módulo UART16550
u8 XUartNs550_RecvByte (u32 BaseAddress)	Recibe un byte por el módulo UART16550
int XUartNs550_Initialize (XUartNs550 *InstancePtr, u16 DeviceId)	Inicializa el dispositivo UART16550 a sus valores por defecto
int XUartNs550_Send (XUartNs550 *InstancePtr, u8 *BufferPtr, unsigned int NumBytes)	Envia cierto número de bytes
int XUartNs550_Recv (XUartNs550 *InstancePtr, u8 *BufferPtr, unsigned int NumBytes)	Recibe cierto número de bytes
int XUartNs550_mIsTransmitEmpty (BaseAddress)	Indica si el buffer del transmisor esta o no vacío
int XUartNs550_mIsReceiveData (BaseAddress)	Indica si el buffer receptor tiene datos
Void XUartNs550_mEnableIntr (BaseAddress) Void XUartNs550_mDisableIntr (BaseAddress)	Habilita y deshabilita la interrupción del módulo respectivamente
void XUartNs550_SetBaud (u32 BaseAddress, u32 InputClockHz, u32 BaudRate)	Coloca el Baud Rate al que trabaja el módulo UART 16550
XUartNs550_mWriteReg (BaseAddress, RegOffset, RegisterValue)	Escribe a registros de configuración
XUartNs550_mReadReg (BaseAddress, RegOffset)	Lee registros de configuración
Función Capa Básica	**Descripción**
void AX_Init()	Inicializa el UART16550 a un Baud Rate
int AXSend_Com(Xuint8 vec[],Xuint8 tam)	Manda vector de comandos al Brazo
int AXSend_Inst(Xuint8 id, Xuint8 inst, Xuint8 param[], Xuint8 sizeparam)	Manda una instrucción en específica al Brazo
int AXRecv_Data(Xuint8 id, Xuint8 param[], Xuint8 sizeparam)	Recibe datos acerca del estado de los motores del Brazo
int AXMov_Base(Xuint16 pos, Xuint8 vel)	Realiza el movimiento de la Base
int AXMov_Join1(Xuint16 pos, Xuint8 vel)	Realiza el movimiento de la unión 1
int AXMov_Join2(Xuint16 pos, Xuint8 vel)	Realiza el movimiento de la unión 2
int AXMov_Wrist(Xuint16 pos, Xuint8 vel)	Realiza el movimiento del Wrist
int AXMov_Gripper(Xuint8 pos, Xuint8 vel)	Realiza el movimiento de las pinzas
Función Capa Intermedia	**Descripción**
int AXPos_Init()	Coloca el Brazo en una posición de reposo
int AXFind_Obj(Xuint8 num, Xuint8 inc,Xuint16 posini)	Hace un tracking del objeto y cuando lo encuentra lo toma
int AXPickup_Base(Xuint16 posB)	Recoge el objeto de la Base externa
int AXLay_Robot(Xuint16 posB)	Coloca el objeto en la Base del Robot
int AXPickup_Robot(Xuint16 posB)	Recoge el objeto de la Base del Robot
int AXLay_Base(Xuint16 posB)	Coloca el objeto en la Base externa

Tabla B.7: Funciones para el brazo manipulador AX-12, incluyendo las capas de controladores, básica e intermedia

B.6. Biblioteca: Co-procesador para imágenes

A continuación se enlistan las funciones software para el control del co-procesador para imágenes, las funciones abarcan distintos niveles de abstracción.

Función Capa Controladores	**Descripción**
int puma_convofpga_sm_0_Write (unsigned int memName, unsigned int addr, unsigned int val);	Escribe datos al co-procesador mediante el bus FSL
int puma_convofpga_sm_0_Read (unsigned int memName, unsigned int addr,unsigned int* val);	Lee datos del co-procesador mediante el bus FSL
int puma_convofpga_sm_0_ArrayWrite (...)	Manda un arreglo de datos
int puma_convofpga_sm_0_ArrayRead(...)	Lee hacia un arreglo de datos
Función Capa Básica	**Descripción**
XStatus Write_ram(Xuint8 *Addr,Xuint32 Ind, Xuint8 Val)	Escribe un valor a la memoria RAM de la tarjeta
Xuint8 Read_ram (Xuint8 *Addr,Xuint32 Ind)	Lee un valor de la memoria RAM de la tarjeta
Función Capa Intermedia	**Descripción**
Xuint16 Get_tam_img()	Obtiene el tamaño de la imagen, este tamaño es mandado por el puerto serie de la PC
XStatus Store_img_ram (Xuint16 f, Xuint16 c)	Va guardando la imagen en RAM conforme le va llegando por el puerto serie
XStatus ReadSend_img_ram (Xuint16 f, Xuint16 c)	Lee la imagen de RAM y la va mandando por el puerto serie

Tabla B.8: Biblioteca de funciones para el co-procesador de imágenes, incluyendo las capas de controladores, básica e intermedia

Apéndice C

Montar Linux en FPGAs

En esta sección se muestran los pasos necesarios para montar un SO tipo Linux en el FPGA. La distribución que se usa para este trabajo es Petalinux de la empresa Petalogix (Petalogix, Agosto 2008). Petalinux es un *port* del SO uClinux especialmente diseñado para trabajar con el procesador Microblaze de Xilinx, aunque también soporta otras arquitecturas como Coldfire, Blackfin y ARM7; además es una distribución gratuita. uClinux es un proyecto destinado a portar Linux a dispositivos sin unidades de manejo de memoria (MMU), esto implica que el espacio de memoria es continuo y fijo, no existe protección de memoria, así que cada proceso del kernel o de usuario puede acceder cualquier espacio de memoria. Aunque las nuevas versiones del procesador Microblaze ya soportan la unidad MMU. Gracias a este proyecto se puede contar con Linux en iPods, celulares, DVD *players*, *VOIP phones*, *web-cams*, procesadores para diferentes aplicaciones embebidas, etc.

Dentro de las ventajas que tiene usar uClinux en el procesador Microblaze es que posee una capa de abstracción entre en software y el hardware ya bien establecida para comunicación con diversos periféricos, es un ambiente de desarrollo ya bien estudiado y familiar para programadores de Linux, es una distribución efectiva en costo (es gratuita); proporciona todo un sistema de archivos, pila de protocolos, *scheduler*, etc.

Existe un concepto importante que se debe tomar en cuenta cuando se realiza un SE, este concepto es el de la "compilación cruzada" (*cross-compilation*), esto se refiere a que el compilador es capaz de crear un código ejecutable para una plataforma distinta a la plataforma donde corre el compilador, este tipo de compiladores generalmente son usados para sistemas empotrados o para sistemas de múltiples plataformas. Es una herramienta que se debe usar para una plataforma en donde no es conveniente o prácticamente imposible compilar en ella, como los micro-controladores que corren con una cantidad mínima de memoria. Todo el desarrollo del sistema es llevado a cabo en una computadora (*Host*) para posteriormente ser programada en otra computadora (*Target*), tal y como lo muestra la figura C.1. La computadora *target* en este caso es un sistema empotrado implementado en un FPGA.

Todas las herramientas que se van a usar para poder montar Linux en el FPGA tienen que estar instaladas en la computadora *Host*, la cual puede tener una de dos configuraciones: la primera de ellas es que tanto las herramientas de Xilinx (EDK, ISE, etc.) como las herramientas para poder compilar la imagen de Linux que correrá en el FPGA, ambas herramientas estén instaladas en un *Host* con SO Linux; la segunda configuración posible y es la que se usa en este trabajo, es que el *Host* con el que se va a trabajar tenga SO diferentes, es decir, que las herramientas de Xilinx se encuentren instaladas en la partición de Windows y las herramientas para compilar la imagen de Linux se encuentren en la partición Linux del *Host*, como lo muestra la figura C.2. En general, para montar un sistema operativo Linux en un FPGA se necesitan cuatro cosas:

1. La plataforma hardware que funciona sobre una tarjeta FPGA.

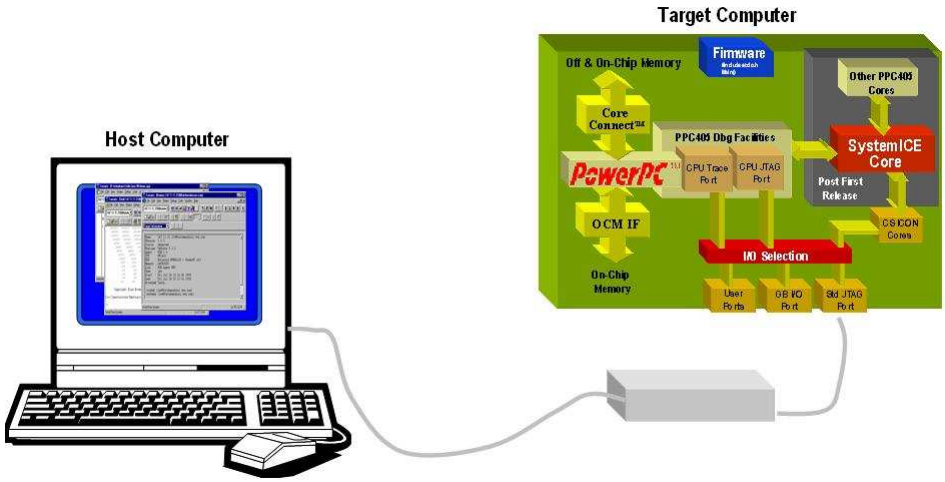


Figura C.1: Concepto de compilación cruzada. La computadora *host* sirve para realizar la compilación de las aplicaciones que correrán en la computadora *target*. Figura tomada de Avnet (Octubre 2007)

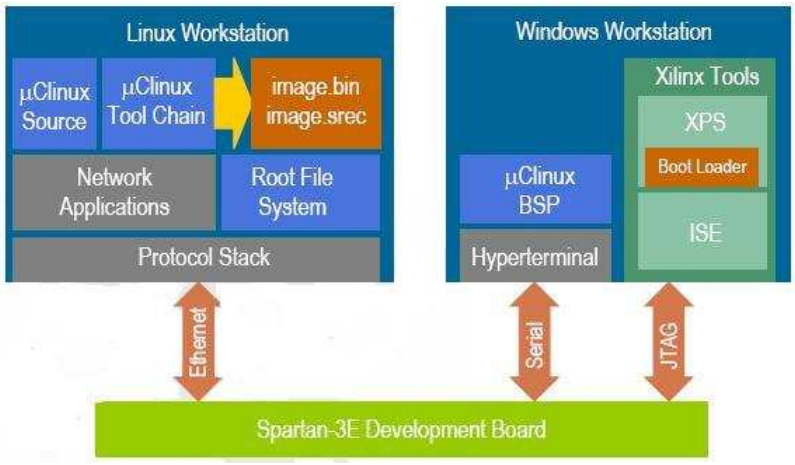


Figura C.2: Configuración Linux-Windows para montar Linux en el FPGA. En la partición de Windows se encuentran las herramientas de desarrollo hardware (i.e. EDK, ISE.) y en la partición Linux se encuentran las herramientas para el desarrollo software (i.e. uClinux, configuración del kernel, aplicaciones de usuario, etc.). Figura tomada de Avnet (Octubre 2007)

2. El BSP necesario para la plataforma hardware.

3. La distribución de Linux, en este caso Petalinux.

4. El MicroBlaze GCC *tool-chain*, el cual incluye todo lo necesario para construir el kernel de uClinux en un ambiente Linux.

En la figura C.3 se observa a grandes rasgos el concepto de tener un SO Linux en un FPGA, tenemos en la capa inferior el hardware del sistema, en este caso la tarjeta FPGA con el sistema empotrado implementado en el dispositivo, después se tiene los controladores para controlar todos los elementos hardware de la plataforma, es decir, el BSP, y finalmente sobre el BSP se

tiene un SO, en este caso uClinux, con todas sus utilidades, bibliotecas, aplicaciones de usuario, etc.

Figura C.3: Diagrama de una arquitectura software con Linux empotrado en un FPGA. El BSP es el que se encarga de la comunicación con los periféricos hardware. Figura basada de Avnet (Octubre 2007)

Una vez que se tiene todos los requerimientos para compilar Linux en el FPGA es necesario seguir un flujo de diseño para lograr esto. En la figura C.4 se observa de forma general el flujo de diseño para correr Petalinux en el FPGA.

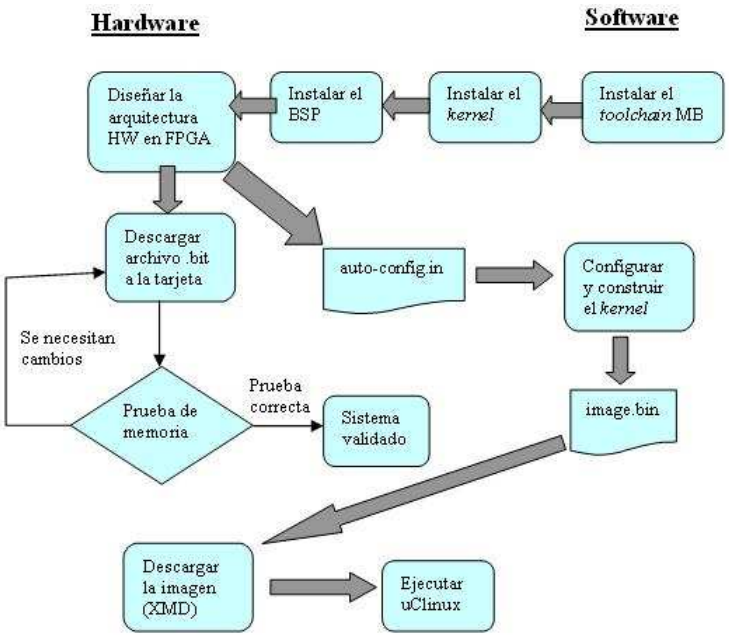


Figura C.4: Flujo de diseño para montar Petalinux en el FPGA. Figura basada de Avnet (Octubre 2007)

En primera instancia tenemos que se debe instalar el MB *tool-chain*, el Kernel de uClinux y el BSP de Petalinux, todas estas fuentes se consiguen en la página oficial del proyecto Petalinux Petalogix, cuando se instala el BSP de Petalinux en las opciones de *Software Platform Settings* (figura A.5) en la pestaña de OS automáticamente aparece la opción de Petalinux como sistema operativo. El siguiente paso es realizar la plataforma Hardware con la herramienta EDK, tal y como se describió en la sección A.1, aquí se realiza también una pequeña aplicación software, generalmente una prueba de memoria y periféricos para verificar que la plataforma hardware funciona correctamente, en este paso cuando se genera el BSP de la plataforma mediante la utilidad LibGen, es generado un archivo importante en el proceso llamado 'Kconfig.auto', el cual contiene toda la información de la plataforma hardware. Este archivo es pasado a la parte Linux de la computadora *Host* para poder compilar la imagen del Petalinux que va a correr el FPGA.

El paso siguiente es generar la imagen Petalinux, así que ahora se tiene que trabajar en la partición Linux de la computadora *Host*. Cuando la distribución Petalinux es instalada, ésta tiene varias herramientas para facilitar un poco la generación de la imagen. Todos los comandos que trae disponibles la distribución Petalinux se tienen que correr en un Terminal de Linux. El primer paso es generar una nueva configuración para la tarjeta que se va a utilizar mediante el *script 'petalinux-new-platform'*, después se tiene que configurar las opciones del kernel con la instrucción *'make menuconfig'*, esta instrucción abre una serie de ventanas en donde el usuario configura todas las opciones del kernel según lo que su sistema requiera, una vez configurado el kernel, se resuelven todas las dependencias y se compila con las instrucciones *'make dep'* y *'make all'* respectivamente, al final del proceso se tiene un archivo binario llamado "image.bin", el cual es una imagen del Linux que va a correr en el FPGA, estos pasos se ilustran en la figura C.5.

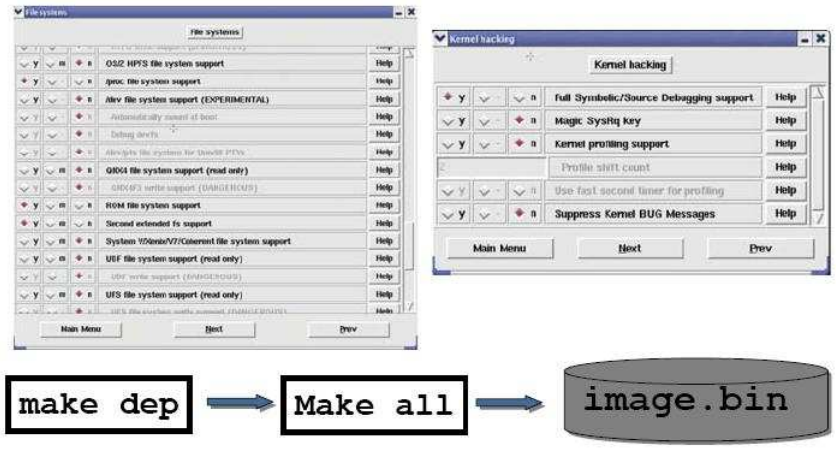


Figura C.5: Generación de la imagen Linux que corre en el FPGA. Figura tomada de Avnet (Octubre 2007)

Una vez que se tiene el archivo image.bin, se pasa nuevamente a la parte Windows de la computadora *Host* para realizar el paso final del flujo de diseño de la figura **??**, esto es, una vez que en el FPGA se tiene programada la parte hardware del sistema, mediante una utilidad de EDK llamada XMD (*Xilinx Microprocessor Debugger*) se baja este archivo a la memoria SDRAM del FPGA. Una vez finalizado este último paso, se necesita algún programa que juegue el papel de terminal para el Petalinux del FPGA, en este caso se usa el Hyperterminal de Windows conectado al periférico DCE de la tarjeta. De esta manera se tiene corriendo un SO Linux dentro del FPGA, tal y como se muestra en la figura C.6.

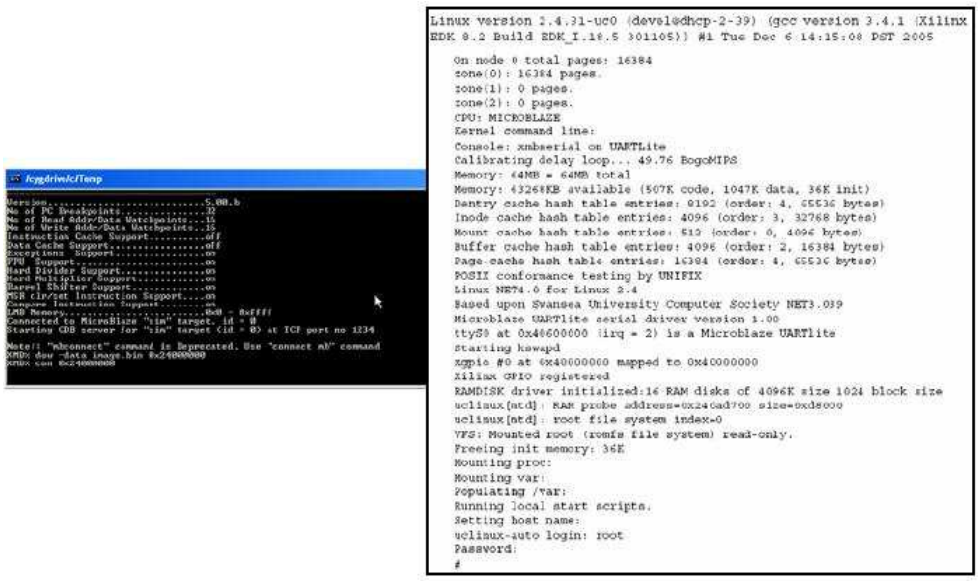


Figura C.6: Proceso de bajar la imagen de Petalinux en el FPGA por medio del XMD de EDK y cómo corre Petalinux en una terminal. Figura tomada de Avnet (Octubre 2007)

Bibliografía

Arias, E. M. & Torres, H. C. 2001, *A Real-time FPGA Architecture for Computer Vision.* Journal of Electronic Imaging (SPIE - IS&T). Vol 10, 289–296

Arimoto, P. 2008, *Proyecto FPGA-based Realtime Vision.* Departamento de Robótica. Universidad de Ritsumeikan, Japón. website: http://www.ritsumei.ac.jp/se/ hirai/research/visionFPGA-e.html Diciembre-2008

Bräunl, T. 2006, *Embedded Robotics: Mobile Robot Design and Applications with Embedded Systems.* Editorial Springer. Segunda Edición

Cañas, J. M., Matellan, V., & Montúfar, R. 2004, *Programación de Robots Móviles.* Revista Iberoamericana de Automática e Informática Industrial (RIAI), ISSN: 1697-7912, Universidad Rey Juan Carlos-España, INAOE-México

Cañas, J. M. & Matellán, V. 2002, *Dynamic schema hierarchies for an autonomous robot.* Advances in Artificial Intelligence (IBERAMIA 2002). Lecture notes in Artificial Intelligence. Editorial Springer. Volume 2527

Don, D., Srinivas, B., & Ranjesh, J. 2005, *Hardware/Software Codesign for platforms FPGA.* Reporte técnico. Xilinx

Dudek, G. & Jenkin, M. 2000, *Computacional Principles of Mobile Robotics.* Cambridge University Press, Primera Edición

Escalera, H. A. 2001, *Visión por computador. Fundamentos y métodos* Universidad Carlos III de Madrid. Editorial Prentice Hall.

González, R. C., Woods, R. E., & Eddins, S. L. 2004, *Digital Image Processing using MATLAB.* Editorial Prentice Hall. Segunda Edición

Kreindler, D. 2008, *Designing and Using FPGAs for Double-Precision Floating-Point Math.* White paper Altera company, http://www.altera.com/literature/wp/wp-01028.pdf, Diciembre-2008

Li, Q. & Lao, C. 2003, *Real Time Concepts for Embedded Systems.* Editorial CMP Books

Malachy, D. 2008, *FPGA Supercomputing Gets Practical for Defense Apps.* Artículo de Cost Journal on Line, http://www.cotsjournalonline.com/home/article.php?id=100408&pg=1, Diciembre-2008.

McBader, S. & Lee, L. 2002, *A 3.23 GOPS Parallel Architecture for Digital Image Pre-Processing.* Proceeding Signal Processing, Pattern Recognition, and Applications (SPPRA 2002)

Muñoz, N. D., Andrade, C., & Ospina, N. L. 2006, *Diseño y construcción de un robot móvil orientado a la enseñanza e investigación.* Revista de la División de Ingeniería de la Universidad del Norte, España. ISSN 0122-3461, No. 19, 114–127

Rodriguez, P. E. & Arias, E. M. 2002, *FPGA Architecture for a Visual Tracking System.* Field Programmable Logic conference (FPL). Montpellier, France, 710–719

Rousseaux, S. 2008, *High Performance Computing on FPGA.* Artículo de la compañía CETIC, http://www.cetic.be/article488.html, Diciembre-2008

Sánchez, B. H. 2008, *Procesador para recuperación 3D apartir del flujo óptico.* Departamento de Ciencias Computacionales. Tesis INAOE

Strenski, D. 2007, *FPGA Floating Point Performance.* Analista de aplicaciones. Artículo de Cray Inc. http://www.hpcwire.com/features/FPGA_Floating_Point_Performance.html. Enero-2007.

Xapp-Xilinx. 2004, *Connecting Curstomized IP to the Microblaze Soft Processor Using the Fast Simplex Link (FSL) Channel.* Artículo de aplicación de la compañía Xilinx ©

Referencias en Internet

Acroname. Diciembre 2008, Acroname website. Disbribuidor de sensores para robótica. *http://www.acroname.com/*

Avnet. Octubre 2007, Avnet uClinux DVD. emphTutorial: Getting Started with uClinux in Spartan 3E Board Kit. *http://www.avnet.com/*

CMUcam. Diciembre 2008, CMUcam website, cámaras para visión en robots. *http://www.cmucams.org*

CrustCrawler. Agosto 2008, CrustCrawler website, fabricante del brazo manipulador *Smart Arm AX-12. http://www.crustcrawler.com*

Evolution. Agosto 2008, Evolution Robotics website, sistema operativo con soluciones modulares para navegación autónoma. *http://www.evolution.com*

Eyebot. Diciembre 2008, Eyebot website, cámara para tareas de visión en robots. , *http://robotics.ee.uwa.edu.au/eyebot*

Gumstix. Diciembre 2008, Gumstix website, tarjeta controladora de prósito general. *http://www.gumstix.org*

HandyBoard. Diciembre 2008, Handy board y handy cricket website, tarjetas controladoras para robots *http://www.handyboard.com*

iRobot. Septiembre 2008, iRobot website, fabricante del robot móvil Create. *http://www.irobot.com*

Kteam. Diciembre 2008, K-team website, fabricante de robots y tarjetas controladoras. *http://www.k-team.com*

Lynxmotion. Diciembre 2008, Lynxmotion website, fabricante de sensores para robótica. *http://www.lynxmotion.com*

MobileRobots. Septiembre 2008, Mobile Robots website, fabricante de los robots PeopleBot y Pioneer. *http://www.mobilerobots.com*

Parallax. Septiembre 2008, Parallax website, sensores y kits para robótica. *http://www.parallax.com*

Petalogix. Agosto 2008, Petalogix website, sitio del SO Petalinux para arquitecturas Microblaze. *http://www.petalogix.com*

PlayerStage. Junio 2008, Player/Stage project website, implementación de un servidor para robots móviles y su simulador. *http://playerstage.sourceforge.net*

RoboticStudio. Junio 2008, Robotics Studio website, herramientas para crear y depurar aplicaciones robóticas. *http://www.microsoft.com/robotics*

Xilinx. Diciembre 2008, Xilinx website, fabricante de FPGAs y herramientas para programarlos como EDK e ISE Foundation. *http://www.xilinx.com*

Xilinx-2. Febrero 2008, Xilinx Company, *EDK Concepts, Tools and Techniques: A Hands-on Guide to Effective Embedded System Design*, Version 10.1i. *http://www.xilinx.com/support/documentation/sw_manuals/edk_ctt.pdf*

Xilinx-3. Diciembre 2008, Xilinx EDK webpage, herramienta para realizar sistemas embebidos en FPGAs. *http://www.xilinx.com/edk*

Xilinx-4. Diciembre 2007, Xilinx Company, *MicroBlaze Processor Reference Guide*, UG081 ver 7.0. *http://www.xilinx.com/microblaze*